Toward an Insider Threat Detection Framework Using Honey Permissions

Parisa Kaghazgaran and Hassan Takabi*
University of North Texas, Denton, TX, USA
parisakaghazgaran@my.unt.edu, takabi@unt.edu

Abstract

The insider threat remains one of the most serious challenges to computer security. An insider attack occurs when an authorized user misuses his privileges and causes damages to the organization. Deception techniques have served as a common solution to insider threat detection, and several techniques, such as approaches based on honey entities, have been proposed. On the other hand, access control systems lack the ability to detect insider threats. In this paper, we focus on integrating deception into the role-based access control (RBAC) model, which is one of the most widely used access control models. We introduce the notion of "honey permission" and use it to extend RBAC to help in insider threat detection. We define honey permissions as permissions that exceed the authorized access, and are assigned to a subset of roles known as "candidate roles". Objects included in honey permissions are fake versions of sensitive objects that are enticing for malicious users. In this way, an attempt to access sensitive resources by unauthorized users would be detected. We extend the RBAC model by adding honey permissions, indicating candidate roles, and adding a monitoring unit which monitors the sessions in which the owners of the sessions activate a subset of candidate roles and have access to an object through a honey permission. We propose an algorithm to select candidate roles and assign honey permissions to them. Furthermore, we provide security analysis and consider the overhead that would be added to the RBAC system for evaluation.

Keywords: Insider Threat, Deception, Role-Based Access Control (RBAC), Honey Permission

1 Introduction

Threats from the inside of an organization are a significant problem, since it is difficult to distinguish them from benign activities. This "insider threat" has received significant attention and is considered one of the most serious security problems [21]. It is also seen as the most difficult problem to deal with, because insiders often have information and capabilities unknown to external attackers, and as a consequence, can cause serious damages. There is not a single agreed upon definition of insider threat. In 2008, a cross-disciplinary workshop on "Countering Insider Threats" concluded that "an insider is a person that has been legitimately empowered with the right to access, represent, or decide about one or more assets of the organization's structure" [5]. Little is known about the insider threat, and the threat of insider activities continues to be of paramount concern to both the public and private sectors [12]. A report recently produced by the security management company "AlgoSec" found that a significant proportion of security professionals view insider threats as their greatest organizational risk [1].

The insider threat has been the subject of extensive studies, and many approaches from technical and behavioral perspective to psychological perspective have been proposed to detect or mitigate insider threat. Most researches haves been done on how to prevent unauthorized and illegitimate access to systems and information, however, most damaging threats are due to internal misuses. In order to form

Journal of Internet Services and Information Security (JISIS), volume: 5, number: 3 (August 2015), pp. 19-36

*Corresponding author: Computer Science and Engineering Department, University of North Texas, 3940 N.Elm St, Denton, TX 76207, Tel: +1-940-565-2385, Web: http://www.cse.unt.edu/~takabi

a clearer definition of insider threats, we distinguish between two types of threats: Masqueraders and Traitors. Masqueraders are illegitimate users who impersonate legitimate users. One standard solution for detection of masqueraders includes anomaly-based IDSs, which try to model normal behavior and detect deviation from that model [18]. Traitors are legitimate users who misuse their privileges to perform an attack. In this paper, we consider insiders as traitors. We note that some outsiders can also become insiders when they obtain internal network access using spy-ware and root-kits [7]. However, our focus is on employees attempting to misuse sensitive information.

The classic and common strategy for preventing insider threats includes access control techniques. These techniques confine the scope of system and information access for any user. In this way, damage from insider threats will diminish. Access control models, such as those proposed by Bell-Lapadula [4] and Clarck-Wilson [9], attempt to limit access to organization resources, but do not provide an explicit solution to the insider threat problem. Although prevention techniques act as a helpful strategy to protect systems and information, they may not always succeed. For this reason, monitoring and detection techniques are required as a second security layer if prevention fails.

Deception has always been a means of detecting or thwarting spying. In particular, the use of decoys (realistic but fake objects) to divert or detect attacks has been proven as a powerful technique. Various deception techniques have been used in computer security to bait attackers with digital decoys like computer files. Decoys, often referred to as honey in computer security, are powerful tools for compromise detection [10]. Honeypots, computer systems deployed to lure attackers, are the best-known example. Many varieties of honey systems have been introduced in the literature, such as honey files, documents, tokens, words, and encryption. We review these techniques in the Related Work section.

We introduce the term "honey permission" and use it to extend RBAC to help in insider threat detection. Honey permissions are permissions that exceed the authorized responsibilities associated with a given role (e.g., access to financial documents for an IT administrator). We focus on RBAC [26], because it is the most widely-used model for advanced access control in diverse enterprises of all sizes, due to its ease of administration and economic benefits. In RBAC, roles represent functions within a given organization. Access permissions are associated with roles instead of users. Users can activate a subset of authorized roles and easily acquire all their permissions.

In RBAC, permission is composed of an object and an operation. The object represents a resource of the system, such as a file, and the operation corresponds to the action that a user can perform on the object, such as read, and write. Since objects are targets of any insider threat, objects included in honey permissions must not be real objects otherwise the system will bear further damages. We assume if a user attempts to access sensitive resources through a honey permission, he could be a potential insider. This then sets off an alarm in the system.

To the best of our knowledge, this is the first work that integrates deception into the access control for insider threat detection. The contributions of this paper include:

- Introducing the notion of honey permission and extending RBAC model for insider threat detection
- Proposing an algorithm that determines:
  - candidate permissions for honey permissions based on their risk
  - candidate roles for honey permissions assignment based on their level of access to system resources

The rest of the paper is organized as follows: Section 2 provides the background knowledge about RBAC. Section 3 describes our proposed approach, which is then analyzed in Section 4. In Section 5, we discuss related work. In Section 6, we conclude the paper and discuss future work.

2 Overview of RBAC

We focus on the NIST RBAC model [26], due to its benefits and ease of administration. RBAC serves as a combination of discretionary and mandatory access control models, which supports organization or user specific requirements [23] [31].

The standard RBAC is organized into two parts: Reference Model and Functional Specifications. The RBAC reference model involves three model components: Core RBAC, Hierarchical RBAC, and Constrained RBAC. Core RBAC includes six basic elements: users (U), roles (R), objects (OBS), operations (OPS), permissions (P) and sessions (S). Roles are a special element of RBAC because they facilitate administrative functions and provide economic benefits for organizations.

In RBAC, permissions are assigned to roles (permission assignment relations), and users are also assigned to roles (user assignment relations). User-role assignment could be done based on trust measure [33]. Roles work as a means to map relationships among users and permissions. Each session is a mapping between a user and an activated subset of authorized roles for that user. Users need to activate a role to obtain its associated permissions in a session. To be more specific, a user establishes a session in which he activates some subset of authorized roles. Each session is associated with a single user, and each user could be associated with one or more sessions.

Role hierarchies, as a key concept of RBAC, define an inheritance relationship among roles. Role r_1 inherits role r_2 if all permissions of r_2 are also permissions of r_1 [25].

Constrained RBAC includes Separation of Duty (SoD) relations. SoD prevents granting two or more conflicting roles to a user. There are several approach to improve separation of duty relations such as using fuzzy relations [32]

Our current approach focuses on Core RBAC and aims to integrate an insider threat detection framework in the standard RBAC in a transparent way so that it can be applied in a border domain. Incorporating this idea in RBAC would add more security to the organization. We name the extended model as Insider Threat Detection Enabled Role-based Access Control (ITD-RBAC) model. A formal definition for standard RBAC model is as follows:

- UA: User Assignment $\subseteq U \times R$
- PA: Permission Assignment $\subseteq P \times R$
- $U - S$: user-sessions $(u : U) \rightarrow 2^S$
- $S - R$: session-roles $(s : S) \rightarrow 2^R$
- $P : 2^{OPS \times OBS}$, the set of permissions
- $Op(p) : (p \in P) \rightarrow Op \in OPS$, returns the operation included in permission p
- $Ob(p) : (p \in P) \rightarrow Ob \in OBS$, returns the object included in permission p

Our definitions of Functional Specifications for ITD-RBAC are based on those already defined in standard RBAC model [26]. Briefly, the Functional Specifications include administrative functions for creation and maintenance of RBAC elements and relations, review functions for performing administrative queries, and supporting system functions for creating and managing RBAC attributes on user sessions and making access control decisions.

3 Proposed Approach

We extend the RBAC model by adding honey permissions, indicating candidate roles, and adding a monitoring unit for insider threat detection. To integrate honey permissions into the system, we focus on the permission assignment relations in the RBAC model. Honey permissions are members of the permissions set associated with a role. The proposed system requirements are as follows:

1. Honey permissions should satisfy these features:
   (a) Permissions with higher risk value are better candidates to be considered for honey permissions (As we will show in section 3.3, permissions including more sensitive objects have higher risk value).
   (b) If a potential insider misuses the honey permission(s), it may cause damage to the system. So, the system should be able to detect suspicious activities of insiders by using the fake version of real objects (honey objects).
   (c) Honey objects must be enticing for potential insiders. Sensitive objects are good candidates for honey objects (As we will show in section 3.1, if sensitive objects are misused the system will bear more damage). The sensitivity level of objects is usually determined by owner of the objects.
2. The system should be able to determine the risk associated with permissions.
3. The system should be able to specify a set of roles as candidate roles for honey permissions. Assigning honey permissions to all the roles is not efficient and may add too much overhead to administrative operations. Therefore, there should be thresholds for number of the honey permissions introduced to the system as well as the number of the candidate roles.
4. It should be possible to perform privilege escalation.
5. If honey permissions are misused an alarm should be generated for insider threat detection.

We extend the RBAC model by adding the following concepts in order to have a new framework for insider threat detection:

- Assigning a value to each object that shows the sensitivity level of the object
- Assigning a value to each permission that shows the risk of the permission
- Assigning a value to each role that shows the risk of the role
- A threshold on risk value that determines which permissions are appropriate for honey permissions
- A threshold determines number of candidate roles for honey permission assignment

For definition of the ITD-RBAC model, we add the following concepts to the standard RBAC definition:

- Candidate Role (R_c) ·
- Honey Permission (HP)
- Honey Permission Assignment relation (HPA).
- Insider Threat Detection Policy ($ITD-Policy$)

We assume that a honey permission is assigned to a role if (1) that role is one of the candidate roles; (2) its risk value is more than the threshold value; (3) its risk value is more than the risk value of associated role. The number of honey permissions assigned to each candidate role is an administrative parameter and regarding system configuration, it varies in different RBAC applications. For the sake of Indistinguishability the number of honey permission assignment relations should be very less than number of regular permission assignment. It means honey permissions must not be distinguishable from regular permissions assigned to a role otherwise potential insiders will refuse requesting honey permissions.

Our approach aims to detect if a user is misusing a honey permission. Therefore, we focus on supporting system functions from Functional Specifications part of RBAC that are required for session management and making access control decisions. An overview of supporting system functions is given below:

- *CreateSession*: Creates a user session and provides the user with a default set of active roles.
- *AddActiveRole*: Adds a role as an active role for the current session.
- *DropActiveRole*: Deletes a role from the active role set for the current session.
- *CheckAccess*: Determines if the session user has permission to perform the requested operation on an object.

The key extensions of our model focus on access decisions. Thus, we redefine the *CheckAccess* function. *CheckAccess* is the authorization decision making function which describes how a decision is made within RBAC model by taking as inputs the current session, the requested operation and the target object. This function returns a Boolean value i.e. true or false.

CheckAccess: $S \times OPS \times OBS \rightarrow BOOLEAN\ Value$

$CheckAccess(s,op,ob) = (\exists r \in R : r \in S - R(s) \bigwedge((op,ob),r) \in PA)$

In general, *CheckAccess* function returns a Boolean value, meaning whether the user of a given session is allowed to perform a given operation on a given object or not. The function is valid if and only if the session identifier is a member of the S data-set, the object is a member of the OBS data-set, and the operation is a member of the OPS data-set. User of a session has permission to perform the operation on that object if and only if that permission is assigned to (at least) one of the active roles of that session.

Otherwise, *AddActiveRole* function will be executed. This function adds a role as an active role, if the requested permission belongs to that role and if user of the session is assigned to that role. When a role is added to the session as an active role, all of its assigned permissions become available in the session.

We do a slight modification in the *CheckAccess* function by adding a statement to check whether the requested permission exists in the honey permission list or not. It means, if the requested permission is assigned to (at least) one of the active roles of that session, it will also be checked against the honey permission list.

Here, we modify the consequences of the results for the *CheckAccess* function. The formal definition is as follows:

False $\rightarrow$Deny access

True $\rightarrow$Check if the access is through a honey permission or not and then allow access.

$CheckHoneyPermission(r,op,ob) = (r \in R_c \wedge (op,ob) \in HP \wedge ((op,ob),r) \in HPA)$

The *CheckHoneyPermission* statement checks if a role r has been defined as a candidate role and the permission has been defined as a honey permission and r has been allocated the honey permission to perform operation op on object ob. If such a value exists, the function returns true, otherwise false.

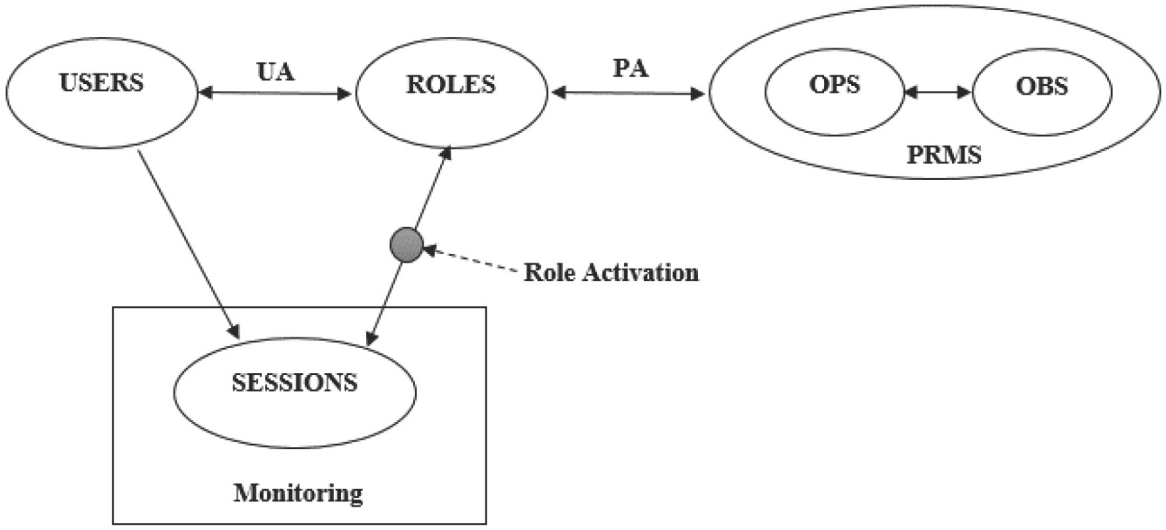

Figure 1: Core RBAC Elements

If it returns true then the monitoring module will be activated and will monitor the object access in that session. After activation of the monitoring module, an alert will be set off based on ITD-Policy.

Generating an alarm immediately after requesting access to honey permission is not acceptable and may cause too many false positive. For example, opening the honey files would not be a good measure for the insider detection. Because users may open such files accidentally. ITD-Policy determines the right time for alarm setting.

ITD-Policy: If honey permission violates the confidentially of a honey object, both opening and spending time on it should be considered. Also, if files are taken away from the organization, it would be a sign of malicious activity. To detect manipulation of objects in need of integrity protection, opening and writing to these objects is considered malicious activity.

Figure 1 shows the extended Core RBAC elements that sessions are surrounded by monitoring module.

The process flowchart for the $ITD-RBAC$ model is shown in Figure 2. $ApplicationService$ provides an interface through which the user can log-in into the host environment. After logging, he can request for an access to the object. The $AuthenticationService$ validates the user's credentials. If the user is not authorized, it denies access to enter the system otherwise he starts Session. The RBAC Policy Check will identify the roles of the user in that session. If the user has a role that is authorized to access the object, the request access is checked through honey permission list and then operation on the object is performed in the host environment. If conditions of $CheckHoneyPermission$ statement are satisfied, then the access for operation on the honey object is granted and the user operation on the honey object is monitored. Otherwise, access request is denied. Monitoring module will do a security check on user behavior log based on $ITD-Policy$ in specific time intervals. If the conditions for generating alert are satisfied then the system will set off an alert to the system admin and user session will be terminated. The time interval is defined by system admin (e.g. every 15 minutes $ITD-Policy$ will be checked). As shown in Figure 2 the only difference between process flow of ITD-RABC and regular RBAC is the checking of an access request against the honey permissions list. If m shows the number of permissions in the system ($|P| = m$), the overhead added in worst case is $O(m)$. That means all the permissions exist

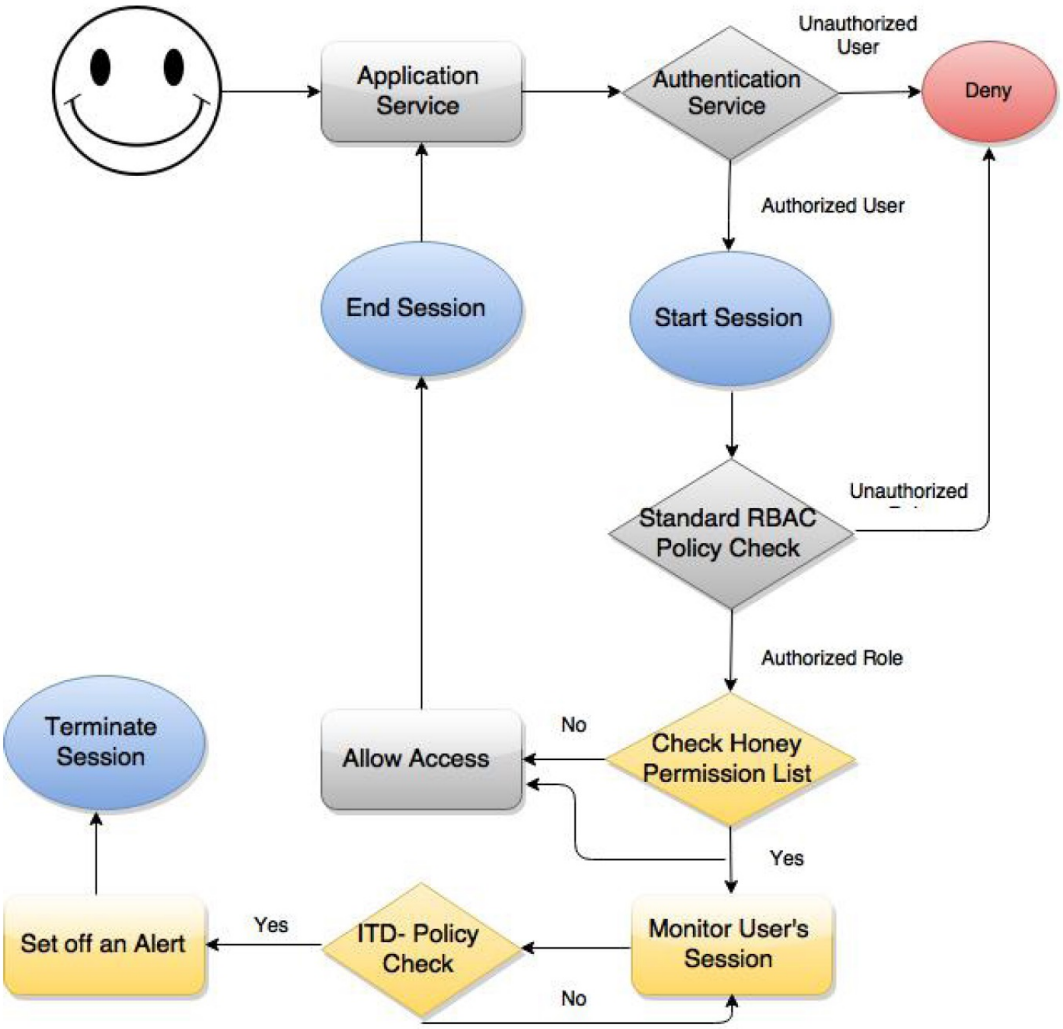

Figure 2: Process Flow Chart of ITD-RBAC Model

in the honey permissions list.

3.1 Sensitivity of Objects

In RBAC, an object can be any system resource, such as: a file, printer, or database records [26]. The purpose of any access control mechanism is to protect objects against loss of confidentiality, loss of integrity, or loss of availability. Because objects are susceptible to different threats, depending on their applications, different objects need different security requirements. For instance, some objects require that their confidentiality be well protected, like medical records and password files. Other objects are sensitive to alternations and need their integrity to remain intact, like network log files. Some objects may be sensitive to both disclosure and alteration.

There are several classifications for data sensitivity. *McLean* classifies data as top secret, secret, confidential, and unclassified [20]. In this paper, we assume that system resources are ordered based on their sensitivity levels and are assigned a sensitivity value. More sensitive objects are more enticing for

insiders.

3.2 Privilege Escalation

The type of operations is dependent on the type of system in which RBAC will be implemented. For example, within a file system, read, write, and execute serve as operations. Within a database management system, operations set includes insert, delete, append, and update. Regarding a file system, read operation affects confidentiality, and write operation affects integrity. We use this notion for privilege escalation in insider detection. Privilege escalation is defined in the following scenarios:

Scenario 1: within a file system, a network administrator role is authorized for permission (network log file, read) and e.g. report malicious activities. If he acts as an insider, he may remove data related to his ID from the log files, or he may add a new record (modifying the log file). While, writing in the log file is not allowed for such a role.

Scenario 2: Password files within an operating system are not readable, but they are written through password change or creating new accounts. Insider detection approach plants readable bogus password files in the file system so that potential insiders can be identified. Privilege escalation provides elevated access to sensitive objects for roles that are authorized to access those objects through another operation.

3.3 Risk of Permissions

Risk is a measure for determining the probability of occurrence of a hazardous situation and its consequences. We define honey permissions based on their risk. $Celikeletal.$ defined risk value of permissions as the probability of occurrence multiplied by the financial loss of the event [8]. In other words, the risk value of a permission p is the probability that is misused multiplied by the consequence damage cost.

In this paper, we look at risk of permissions from different viewpoint. Since in our extended RBAC model honey permissions are directly assigned to the roles, the probability of occurrence would be 1. So, we only consider damage cost to calculate the risk value of permissions.

$$risk(p) = \sum_{x_p \in MaliciousUsage} cost(x_p) \tag{1}$$

Malicious Usage is a set of possible events that can lead to a misuse of $object$ through operation and $cost(x_p)$ is its associated cost [3].

Scenario 3: Consider a financial institution like a bank and permission $p = (customer-accounts, read)$. In order to calculate the risk of p, suppose the customer account file is confidential, and its leakage leads to 40,000 dollar financial loss. According to system configuration, the probability of occurrence of the event through honey permission is 1. So, risk value of the permission p is supposed to be 40,000 same as damage cost.

In this paper, we consider permissions with high-risk value as the good candidates for honey permissions. As long as permission has more risk value, it is more likely to be misused.

3.4 Candidate Roles

Our assumption is that the users assigned to roles involving high-risk permissions have a higher potential of misusing their permissions and performing malicious activities. Scenario 4 exemplifies this assumption:

Scenario 4: Considering the role structure for a bank, employees work as members of the "employee" role. Above this role may be roles such as "department manager" and "accountant" which inherit all permissions of the "employee" role in addition to their direct assigned permissions. Above

"department manager" could be "saving manager", "loan manager" and so on. Therefore, if a role is placed in top levels of a role hierarchy, it includes not only its direct permissions, but also permissions from its descendants.

Indeed, direct-assigned permissions have more risk value than those inherited from role hierarchy. Therefore, roles located at the top level of role hierarchy are more likely to obtain honey permissions. In our approach, honey permissions are mostly assigned to roles involving high-risk permissions to detect if they misuse their privileges.

We also consider a risk value for each role based on its authorized permissions to determine candidate roles. The risk of a role is a function of the risk of permissions that can be accessed through that role [3]. In this paper, we utilize Root Mean Square (RMS) measure to calculate the risk associated with each role. The average of the risk of permissions is not a good measure to represent the risk of the role, because, in cases that a role owns a few permissions with high-risk values and several permissions with low or medium risk values, the average would be low. Also, the maximum risk value of permissions assigning to a role is not a precise representative for risk of that role. If a role (r_1) has only one or a few high-risk permissions and many low-risk permissions, then risk of the role, based on maximum risk value of permissions, will be high. On the other hand, suppose all or most of the permissions in another role (r_2) have average-risk and high-risk, but maximum risk value of r_2 is less than maximum risk value of r_1. So, the risk of r_1 is larger than the risk of r_2 when r_2 has access to more high-risk permissions.

RMS is a statistical measurement of the magnitude of varying quantity [19]. In the case of a set of n values $\{x_1, x_2, \ldots, x_n\}$, the RMS is calculated using equation 2:

$$\sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)/n} \tag{2}$$

For example, role r involves four permissions p_1, p_2, p_3, and p_4 with risk values of 80, 10, 20, and 25. The average of risk values is 33.75, while RMS is 43.37.

According to the configuration of the system, algorithm 1 selects k roles with the highest risk value among existing roles as the candidate roles for assigning honey permissions.

3.5 Honey Permission Assignment Algorithm

In this section, we propose an algorithm that describes how honey permissions are assigned to candidate roles. Risks are assigned based on the damage cost, in case permissions are misused. In this paper, misusing a permission means having access to objects that users are not authorized for based on RBAC rules.

We define a threshold for the risk of honey permissions. If risk of any permission is more than the threshold, it will be considered as a honey permission. The algorithm will replace its objects with a fake object and then add it to honey permissions set (lines 2 to 5).

After that, the algorithm calculates risk of roles to select candidate roles. This calculation is based on equation 2 (RMS) using risk value of permissions associated with a role. We also define a threshold for the risk of candidate roles, and if risk value of a role is more than the threshold, it will be added to candidate roles set (line 6 to 8).

For each role in the candidate roles set, honey permissions with the risk greater than risk of the role are assigned to the candidate role. Also, numbers of users that are authorized for each candidate role are counted (lines 9 to 13). Number of users is a measure to evaluate how many users have access to honey permissions. We suppose, if the monitoring module can monitor more sessions of the users, then more potential insiders are monitored based on their access requests, thereby increasing the detection rate.

The computational complexity of the proposed algorithm depends on the number of candidate roles and the number of honey permissions to be assigned to each candidate role. The number of candidate

Algorithm 1 Honey Permission Assignment

Require: $P(r)$: returns permissions assigned to role r
$U(r)$: returns users assigned to role r
$U_{insider}$: set of users authorized for honey permissions= null
$Create-fake(Ob(p))$: create a fake version of the object included in permission p
$Risk-\{role\}(r)$: returns risk value of role r
$Risk-\{permission\}(p)$: returns risk value of permission p
θ_p: threshold for risk value of permissions
θ_r: threshold for risk value of roles
k: number of honey permissions assigned to each candidate role

1: **for all** $p \in P$ **do**
2: **if** $Risk-\{permission\}(p) \geq \theta_p$ **then**
3: $o' = Create-fake(Ob(p))$
4: $hp = \{o', Op(p)\}$
5: $HP = HP + hp$
6: **end if**
7: **end for**
8: **for all** $r \in R$ **do**
9: **if** $Risk-\{role\}(r) \geq \theta_r$ **then**
10: $R_c = R_c + r$
11: **end if**
12: **end for**
13: **for all** $r \in R_c$ **do**
14: Assign k number of honey permission to r
15: **if** $Risk-\{permission\}(hp) > Risk-\{role\}(r)$ **then**
16: $P(r) = P(r) + hp$
17: $U_{insider} = U_{insider} + |U_{(r)}|$
18: **end if**
19: **end for**

roles in the worst case equals the number of roles in the system (n). If we define m number of honey permissions, in the worst case, all honey permissions are assigned to each candidate role, so the cost would be $O(n \times m)$. Honey permission assignment is part of administrative functions. Therefore, we should define a new administrative function named "GrantHoneyPermission" in which Algorithm 1 should be executed.

4 Evaluation and Security Analysis

In this section, we evaluate the proposed approach by performing experiments to analyze the overhead introduced to the system. We also provide a security analysis of the proposed approach.

4.1 Evaluation and Experimental Results

A big challenge in evaluating insider threat detection approaches is that no standard metrics or methods exist for measuring success in reducing insider threats [13]. Another challenge is the lack of appropriate data and "ground truth" for evaluating detection performance. These challenges are exacerbated by

the large degree of overlap between observable or measurable behaviors associated with normal versus malicious activities. To evaluate effectiveness of the proposed approach, we focus on the overhead it introduces to an RBAC system instead of accuracy of insider detection. The reason for this is that in our proposed ITD-RBAC, detection rate depends on how well the honey permissions are defined and assuming they are defined properly, based on ITD-Policy, if a user misuses his honey permissions he will be detected as a potential insider. Therefore, we analyze the overhead and complexity of the extended RBAC model.

Our goal is to integrate honey permissions in RBAC in the most efficient way and with minimum effort and minimum overhead introduced to the system. To evaluate the proposed approach, we need to have a measure to calculate the overhead introduced into an RBAC system.

A standard RBAC state is defined as (R, UA, PA, RH). We define a new RBAC state that includes honey permissions and its complexity shows the overhead introduced to the system. R, UA and PA represent roles, user assignment relation and permission assignment relation respectively. $RH \subseteq R \times R$ is a partial order over R, which is called role hierarchy.

A honey permission assignment is shown as (R_c, HP, HPA), where R_c is the set of candidate roles, HP is the set of honey permissions, and $HPA \subseteq R_c \times HP$ is the set of role-honey permission assignment relation.

There are several metrics that could be used to measure the goodness of our proposed RBAC system at detecting insiders with minimum effort. One approach would be to minimize the number of candidate roles [34] [14]. Another approach would be to minimize the number of honey permission-role assignment relations. Also, we can consider how to minimize the administration cost of the resulting RBAC model.

We use weighted structural complexity [22], which is the most general and most flexible measure for goodness of an RBAC state.

Mollyetal. in [22] proposed the notion of weighted structural complexity (WSC) that sums up the number of relationships in an RBAC state, with adjustable weights for different kinds of relationships. The WSC of an RBAC state y, denoted as (y, W), is computed based on equation 3:

$$wsc(y, W) = w_r \times |R| + w_u \times |UA| + w_p \times |PA| + w_h \times |RH| \tag{3}$$

Where $|\ .\ |$ indicates the size of the set or relation.

In order to have a measure, we adopt the notion of WSC as a measurement of goodness of an RBAC state. It can cover other measures and considers how many honey permissions are introduced to the system, how many candidate roles are selected, and number of honey permission assignments among other items.

In our approach, R, UA, and RH are invariant in comparison to standard RBAC, so we only need to evaluate the new RBAC state using varying parameters. We do not add any new role to the system; we select some of the existing roles as candidate role; looking at original WSC, we only need to evaluate HP and HPA for ITD-RBAC.

The WSC in ITD-RBAC is formally defined as follows:

Definition 1. Given $W = < Whp, Whpa >$, where Whp, Wpa represent weights of honey permissions (HP), and honey permission assignment relations (HPA) respectively.

$$wsc'(y, W) = w_{hp} \times |HP| + w_{hpa} \times |HPA| \tag{4}$$

It is possible to adjust the weights of WSC to evaluate the complexity added by honey permission assignment. We consider same weight for all parameters and calculate WSC for different number of these parameters.

4.1.1 Experimental Results

We perform experiments to calculate the WSC for the proposed ITD-RABC based on honey permissions added to the system. To do that, we have implemented our proposed algorithm and use a data-set that has previously been used in the literature. *Ene at el*. obtained two data-sets from Cisco firewalls (Americas small and Americas large) that authenticate external users and provide them with limited network access based on their user profiles [11]. We use "Americas Large" data-set to run the algorithm on it. The data-set contains 404 roles, 3,485 users, 10,127 permissions, 3,965 user assignment relations, 85,508 permission assignment relations and number of role hierarchy relations is 266. The WSC for standard RBAC is 404+3,965+85,508+266=90,143.

The risk assigned to each of the permissions is a random value in the interval [0,100], using uniform distribution. Risk of roles is calculated using RMS formula (Equation 2).

Since we do not have any knowledge about the sensitivity of objects, risk values of permissions are assigned randomly. Since the risk values of permissions are assigned randomly, the risk values of the roles are random too. In the real world, however, roles that are placed at the top of role hierarchy are authorized for higher-risk permissions.

As expected, the number of users that should be monitored increases when number of candidate roles increases. Table 1 through 3 show the relation between WSC and number of monitored users. Therefore, if we want to monitor more users for insider threat detection then the overhead introduced to the system will increase.

Table 1: WSC for $\theta_p = 99$

θ_r	R_c	$\|HP\|$	$\|HPA\|$	$\|U_{insider}\|$	$WSC, w = \{1,1\}$
80	3	127	30	5	157
70	7	127	70	12	197
60	66	127	660	208	787
55	335	127	3350	3431	3477
50	385	127	3850	3480	3977

Table 2: WSC for $\theta_p = 90$

θ_r	R_c	$\|HP\|$	$\|HPA\|$	$\|U_{insider}\|$	$WSC, w = \{1,1\}$
80	3	977	30	5	1007
70	7	977	70	12	1047
60	66	977	660	208	1637
55	335	977	3350	3431	4327
50	385	977	3850	3480	4827

Table 3: WSC for $\theta_p = 85$

θ_r	R_c	$\|HP\|$	$\|HPA\|$	$\|U_{insider}\|$	$WSC, w = \{1,1\}$
80	3	1527	30	5	1557
70	7	1527	70	12	1597
60	66	1527	660	208	2187
55	335	1527	3350	3431	4877
50	385	1527	3850	3480	5377

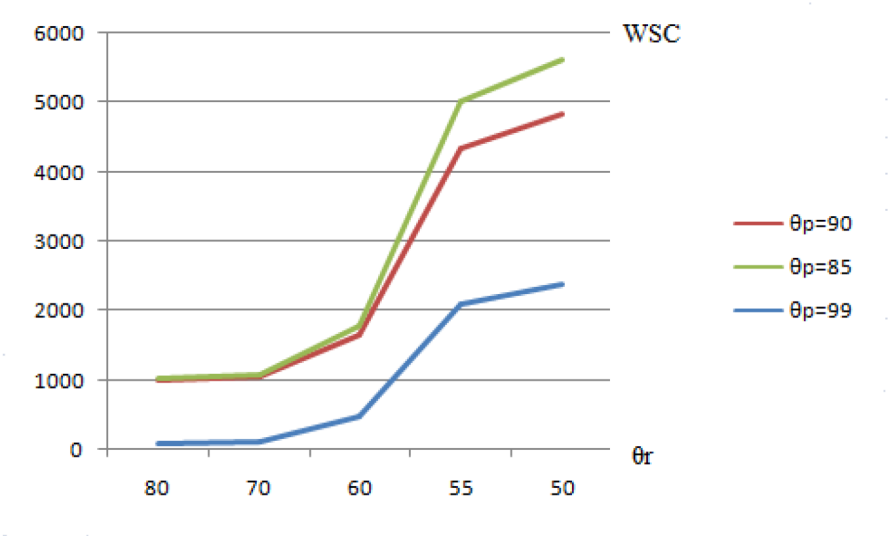

Figure 3: The relationship between θ_r and WSC

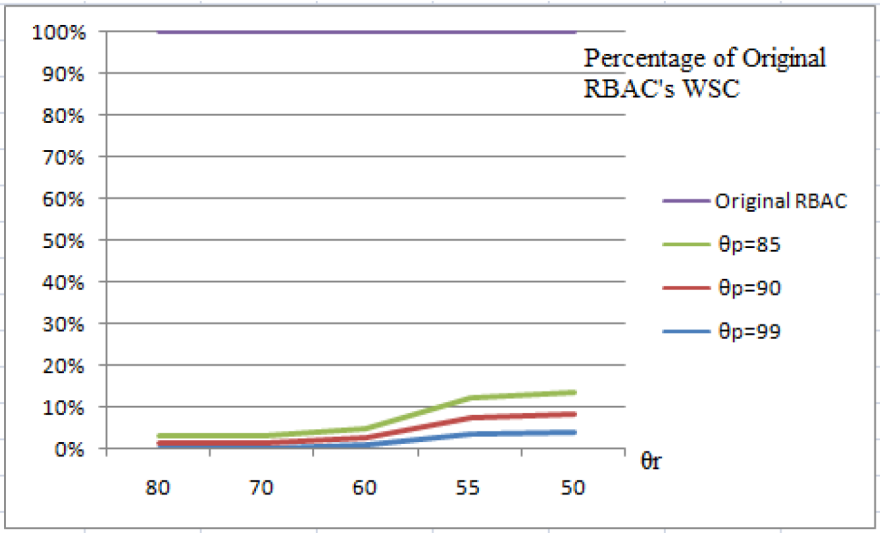

Figure 4: The Percentage of Overhead Introduced by ITD-RBAC with Different Value of θ_r

Figure 3 shows the relation between θ_r and WSC. While θ_r decreases we will have more number of candidate roles and more number of honey permission assignment relations subsequently, as a result, WSC increases. When θ_p decreases there will be more number of honey permissions, as a result WSC increases. As it is shown in figure 4, the overhead introduced by ITD-RBAC is negligible in comparison with original RBAC's WSC. It confirms the efficiency of our approach while it provides more security in original RBAC by detecting insider threat.

4.2 Security Analysis

Ari Juels provided two properties for honey objects: Indistinguishability and secrecy [15]. In this section, we extend notion of indistinguishability and secrecy for honey permissions. The following definition is adopted from [15]:

"Consider a simple system in which $S = \{s_1, ..., s_n\}$ denotes a set of n objects of which one, $s^* = s_j$ for $j \in \{1, ..., n\}$, is the true object, while the other $n-1$ are honey objects."

Regarding this definition, *Ari Juels* described these properties as follows:

1. Indistinguishability: From the attacker's point of view, it must be hard to distinguish honey objects from real objects. Formally, honey objects should be produced from a probability distribution similar to the probability from which a real object s^* was selected.

2. Secrecy: j should be secret. A system with honey objects can only deceive attackers who do not know j, so j cannot be placed among S. based on *Kerckoff*'s principle, the distinction between honey objects and real objects must remain secret, not the fact of using honey objects.

In this paper, we propose an approach in which access control systems support honey systems. We use the honey permissions to deceive insiders rather than focusing on honey objects. In our approach, for a given role that is considered a candidate role, a list of n permissions ($P = \{p_1, p_2, \ldots, p_n\}$) is stored. In this list, $p* = p_j$ is a honey permission, while for $i \neq j$ is the authorized permission of the role. When a user request to access to a specific permission in the system, the access request is checked against list P. If $p = p_i$ for some $i \neq j$, then the access request proceeds normally. If, however, p is a honey permission, $p = p_j$, the monitoring module will monitor the user's session.

Therefore, Indistinguishability means the inability of an insider to determine whether the permission is assigned legitimately to a role or it is a honey permission. We assign honey permissions to candidate roles based on their risk value. The more risk value the better honey permission. We assume the users are aware of the risk value of permissions. If risk of permission p_j is considerably more than risk values of other permissions in list P (authorized ones), then p_j does not satisfy Indistinguishability and can be distinguished easily. Thus, Indistinguishability in this setting means that risk of p_j and the authorized permissions of the role are close. Also, number of honey permissions assigned to a candidate role should be very less than number of authorized permissions; otherwise, the user can distinguish honey permission with high probability.

Secrecy means the set of honey permissions should be secret. For example, we consider 127 honey permissions for $\theta_p = 99$ in the experiment. In a real-world system, the list of honey permissions is stored in Policy Administration Point (PAP). This set should remain secret to insiders.

Bowen et al. enumerate some specific properties for honey objects that are used to entice insiders. These features are believable, enticing, conspicuous, detectable, variability, non-interference, differentiable [6]. Readers are referred to this paper for detail of each feature. In this paper, we assume that honey objects used in honey permissions satisfying these properties.

5 Related Work

Deception technologies play a valuable role in insider detection. *Almeshekah et al*. present a model that shows how to plan and integrate deception in computer security defenses [2].

The first use of decoys to catch insiders was introduced by *Cliff Stoll* [30]. He created "bait" files, which contained non-sensitive information and embedded alarms so that, if anyone accessed these files, he would know. His work uses deceptive resources to attract and detect adversaries. These techniques are commonly known as Honeypots.

There have been several research efforts for information gathering and detection of outsider attacks using Honeypots [28]. Honeypots serve as fake systems for collecting information and understanding how attackers behave. While honeypots have been utilized to detect malicious attackers, *Spitzner* discusses how honeypot technologies can be used to detect insider attacks [27]. He also introduces "honey tokens" as fake medical records, credit card numbers, and credentials in [29].

Yuill et al. introduce a "honey file" system by extending the notion of "honey tokens" to creating bait files [36]. Their system works like an enhanced Network File Server. Any file in the system could be a candidate for a "honey file", and a record will associate a file-name to each user-id. The "honey file" system monitors all file access and generates alarms if a user accesses the "honey files". It is an intrusion detection system that can detect attackers who are connected to the file server via network.

Most of the previous works focus on how to use bait files to detect insiders and rarely pay attention to content or automatic creation of files. *Bowen et al*. introduce a set of properties for decoys to improve their design and maximize the deception they provide for different insiders based on their level of knowledge [6]. They present the D3 (Decoy Document Distributor) system that generates and stores decoy documents on OS file system automatically. To detect when and where a specific decoy is opened, stealthy beacons are watermarked in documents header to send a signal to a server. They evaluate their approach on honeypots.

Most of the previous approaches for insider identification utilize forensic analysis after an attack, rather than using technologies that prevent, detect, and deter insider attacks. *Baracaldo et al*. incorporate the RBAC model with a risk-assessment process and the trust the system has in its users to prevent insider attack [3].

Recently, several new techniques have been developed to deceive insiders and detect them before irrecoverable damages. *J. Voris et al*. focus on foreign language decoy documents that contain enticing proper nouns, such as company names, excited topics, or log-in information [35]. Such information is untranslatable, so they are readable for users who are unfamiliar with that foreign language. Their goal follows three main purposes: first, using a language that is not common in normal business practice; second, the attacker will need to take away the document to translate it; third, authors consume adversarial resources as attackers must still read the document and decide if it contains valuable information, since it will be somewhat scrambled through translation.

Y.park et al. propose a software-based decoy system to deceive insiders who take away dedicated source code. The system generates believable Java source code that appears to an adversary to be entirely valuable software. So, bogus software is generated iteratively using code obfuscation techniques to transform original software. Beacons are injected into the bogus software to detect the taking away; and, if the decoy software is accessed, compiled, or executed, it will generate an alert [24].

Arie Juels also uses the problem of weak passwords to introduce a honey encryption (HE) technique [16]. HE creates cipher texts that, decryption of them by an incorrect key or password, leads to a valid-looking but bogus messages. Attackers then cannot recognize whether decryption has been successful or not.

In another work, *Arie Juels et al*. suggest a method for increasing the security of hashed passwords [17]. For this purpose, false passwords named "honey words" are maintained for every user account. When an adversary achieves a hashed password file and inverts the hash function, he will not recognize whether he has found the true password or a "honey word". If he uses "honey word" to log in to the system, then an alarm will be set off.

In summary, honeypots are the best known example of honey items. However, many types of honey systems have been proposed, such as honey files, honey documents, honey tokens, honey encryption, and honey words. In this paper, we introduced honey techniques in the RBAC access control model known as honey permissions.

6 Conclusion and Future Work

In this paper, we proposed a new approach to detect insider threats by integrating deception techniques into an access control model. Building on top of existing deception techniques using honey entities, we

introduced notion of honey permission and extended the RBAC model to detect insider threats. In order to prevent damages to the organization, we create fake versions of sensitive resources to be included in honey permissions. We proposed algorithms to generate honey permissions based on their risk exposure and assign them to candidate roles. We also performed experiments to evaluate the proposed algorithm. Our results show that the proposed approach can detect insider threats with minimum effort and by adding only a reasonable overhead to the RBAC model.

For future work, we plan to design a prototype for ITD-RBAC to show the usability of our approach. We also plan to integrate honey permissions into other access control models, such as attribute based access control model.

References

[1] AlgoSec. The state of network security 2013: Attitudes and opinions. `http://www.algosec.com/resources/files/Specials/Survey%20files/State%20of%20Network%20Security%202013_Final%20Report.pdf`, 2013.

[2] M. H. Almeshekah and E. H. Spafford. Planning and integrating deception into computer security defenses. In *Proc. of the 2014 workshop on New Security Paradigms Workshop (NSPW'14), Victoria, BC, Canada*, pages 127–138. ACM, September 2014.

[3] N. Baracaldo and J. Joshi. A trust-and-risk aware RBAC framework: tackling insider threat. In *Proc. of the 17th ACM Symposium on Access Control Models and Technologies (SACMAT'12), Rutgers University, Newark, USA*, pages 167–176. ACM, June 2012.

[4] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report 2547, MITRE, March 1973.

[5] M. Bishop, D. Gollmann, J. Hunker, and C. W. Probst. Countering insider threats. In *Proc. of Dagstuhl Seminar, Dagstuhl, Germany*, volume 08302, pages 1–18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, July 2008.

[6] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo. Baiting inside attackers using decoy documents. In *Proc. of the 5th International ICST Conference on Security and Privacy in Communication Networks (SecureComm'09), Athens, Greece, LNICS*, volume 19, pages 51–70. Springer Berlin Heidelberg, September 2009.

[7] J. Butler and S. Sherri. Security: Spyware and Rootkits. 29(6):8–15, December 2004.

[8] E. Celikela, M. Kantarcioglua, B. Thuraisingham, and E. Bertinob. A risk management approach to RBAC. *Risk and Decision Analysis*, 1(1):21–33, 2009.

[9] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proc. of the 1987 IEEE Symposium on Security and Privacy (S&P'87), Oakland, California, USA*, pages 184–184. IEEE, April 1987.

[10] F. Cohen. The use of deception techniques: Honeypots and decoys. *Handbook of Information Security*, 3:646–655, 2006.

[11] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proc. of the 13th ACM Symposium on Access Control Models and Technologies (SACMA'08), Estes Park, CO, USA*, pages 1–10. ACM, June 2008.

[12] P. Fleeger and P. Charles. Reflections on the insider threat. In S. J. Stolfo, S. M. Bellovin, A. D. Keromytis, S. Hershkop, S. W. Smith, and S. Sinclair, editors, *Insider Attack and Cyber Security*, volume 39, pages 5–16. Springer US, 2008.

[13] F. L. Greitzer, T. Ferryman, et al. Methods and metrics for evaluating analytic insider threat tools. In *Proc. of the 2013 IEEE Security and Privacy Workshops (SPW'13), California, USA*, pages 90–97. IEEE, May 2013.

[14] J. H. Jafarian, H. Takabi, H. Touati, E. Hesamifard, and M. Shehab. Towards a general framework for optimal role mining: A constraint satisfaction approach. In *Proc. of the 20th ACM Symposium on Access Control Models and Technologies (SACMAT'15), Vienna, Austria*, pages 211–220. ACM, June 2015.

[15] A. Juels. A bodyguard of lies: the use of honey objects in information security. In *Proc. of the 19th ACM Symposium on Access Control Models and Technologies (SACMA'14), Ontario, Canada*, pages 1–4. ACM, June 2014.

[16] A. Juels and T. Ristenpart. Honey encryption: Encryption beyond the brute-force barrier. *IEEE Security & Privacy*, 12(4):59–62, 2014.

[17] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS'13), Berlin, Germany*, pages 145–160. ACM, November 2013.

[18] P. Kaghazgaran and B. Sadeghiyan. Masquerade detection using gui events in windows systems. *INTERNATIONAL JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY*, 3:67–71, March 2011.

[19] N. Levinson. The Wiener RMS (root mean square) error criterion in filter design and prediction. pages 129–148, 1947.

[20] J. McLean. The specification and modeling of computer security. *IEEE Computer*, 23(1):9–16, 1990.

[21] K. Mickelberg, N. Pollard, and L. Schive. US Cybercrime: Rising Risks, Reduced Readiness Key Findings from the 2014 US State of Cybercrime Survey. `http://www.pwc.com/en_US/us/increasing-it-effectiveness/publications/assets/pwc-2014-us-state-of-cybercrime.pdf`, June 2014.

[22] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo. Evaluating role mining algorithms. In *Proc. of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT'09), Stresa, Italy*, pages 95–104. ACM, June 2009.

[23] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.

[24] Y. Park and S. J. Stolfo. Software decoys for insider threat. In *Proc. of the 7th ACM Symposium on Information, Computer and Communications Security (AsiaCCS'12), Seoul, Korea*, pages 93–94. ACM, May 2012.

[25] R. Sandhu. Role activation hierarchies. In *Proc. of the 3rd ACM workshop on Role-based Access Control (RBAC'98), Fairfax, VA, USA*, pages 33–40. ACM, October 1998.

[26] R. Sandhu, D. Ferraiolo, and R. Kuhn. Role based access control. ANSI INCITS 359-2004, 2004. American National Standards Institute.

[27] L. Spitzner. Honeypots: Catching the insider threat. In *Proc. of the 19th Annual Computer Security Applications Conference (ACSAC'03), Las Vegas, NV, USA*, pages 170–179. IEEE, December 2003.

[28] L. Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[29] L. Spitzner. Honeytokens: The other honeypot. `http://www.symantec.com/connect/articles/honeytokens-other-honeypot`, 2003.

[30] C. Stoll. *The cuckoo's egg: tracking a spy through the maze of computer espionage*. Pocket Books, 2005.

[31] H. Takabi, M. Amini, and R. Jalili. Enhancing role-based access control model through fuzzy relations. In *Proc. of the 3rd International Symposium on Information Assurance and Security (IAS'07), Manchester, UK*, pages 131–136. IEEE, August 2007.

[32] H. Takabi, M. Amini, and R. Jalili. Separation of duty in role-based access control model through fuzzy relations. In *Proc. of the 3rd International Symposium on Information Assurance and Security (IAS'07), Manchester, UK*, pages 125–130. IEEE, August 2007.

[33] H. Takabi, M. Amini, and R. Jalili. Trust-based user-role assignment in role-based access control. In *Proc. of the 2007 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA'07), Amman, Jordan*, pages 807–814. IEEE, May 2007.

[34] H. Takabi and J. B. Joshi. StateMiner: an efficient similarity-based approach for optimal mining of role hierarchy. In *Proc. of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT'10), Pittsburgh, USA*, pages 55–64. ACM, June 2010.

[35] J. Voris, N. Boggs, and S. J. Stolfo. Lost in translation: Improving decoy documents via automated translation. In *Proc. of the 2012 IEEE Symposium on Security and Privacy Workshops (SPW'12), San Francisco,*

California, USA, pages 129–133. IEEE, May 2012.

[36] J. Yuill, M. Zappe, D. Denning, and F. Feer. Honeyfiles: deceptive files for intrusion detection. In *Proc. of the 2004 Information Assurance Workshop (from the 5th Annual IEEE SMC), The Hague, The Netherlands*, pages 116–122. IEEE, October 2004.

Author Biography

Parisa Kaghazgaran is Ph.D. student in Computer Science and Engineering at the University of North Texas. Before coming to US she worked as a research assistant on Digital Forensics and Malware Analysis (2011-2014) in an affiliated research center at Amirkabir University of Technology. Before that, she received a BS in Computer Science (2009) and an MS in Information Security (2011) from Amirkabir University of Technology, Tehran, Iran. She has done a broad research on security topics such as IDS, Privacy, and Insider Threat. She is student member of IEEE. Contact her at parisakaghazgaran@my.unt.edu.

Hassan Takabi is an Assistant Professor of Computer Science and Engineering at the University of North Texas, Denton, Texas, USA. He is director and founder of the INformation Security and Privacy: Interdisciplinary Research and Education (INSPIRE) Lab and a member of the Center for Information and Computer Security (CICS), which is designated as National Center for Academic Excellence in Information Assurance Research (CAE-R) and Education (CAE-IAE). His research is focused on various aspects of cybersecurity and privacy including advanced access control models, insider threats, cloud computing security, mobile privacy, privacy and security of online social networks, and usable security and privacy. He is member of ACM and IEEE. Contact him at takabi@unt.edu.