Analysis of Privilege Escalation Based on Hierarchical RBAC Model

Li Ma, Yongjie Yan, Huowen Jiang and Yanjie Zhou

School of Mathematics and Computer Science, Jiangxi Science & Technology Normal University, Nanchang, 330038, China
Email: liima@sina.com

Abstract: As access control policies become more and more complex, the detection of access control vulnerabilities becomes more important. Previous research efforts have concentrated on access control vulnerabilities due to programming errors, while the privilege escalation caused by logic errors or abuse of privileges has seldom attracted researchers’ attention, which is also a kind of access control vulnerabilities. To investigate the property of privilege escalation, hierarchical RBAC model is used to describe complex relations between different roles that are represented by a directed role graph. Permissions are divided into multiple categories according to the inheritance of permissions in the role hierarchy. Three types of vulnerabilities, Upward Privilege Escalation, Downward Privilege Escalation and Horizontal Privilege Escalation, are defined and decided theoretically based on the inheritance relations between roles in a role graph. Besides the three type, another type of privilege escalation that is not related to the hierarchy of roles is also studied. Finally, the decision theorems of three vulnerabilities are used to optimize the access control decision algorithm.

1. Introduction

New complex computing, such as cloud computing and Internet of Things, puts forward new requirements on access control technology to support fine-grained, dynamic authorization, privacy protection, etc., making people shift their research interests from traditional access control to more complex access control models [1]. However, the complication of access control technology has also made access control vulnerabilities more complicated. For example, the statistical ranking of access control vulnerabilities in Top Ten increased from 7th in 2013 [2] to 5th in 2017 [3] announced by OWASP (Open Web Application Security) organizations, which showed that the harm caused by access control vulnerabilities to various applications cannot be ignored. In the past, researchers generally believed that access control vulnerabilities were caused by developers who did not understand access control knowledge [6] or make errors in technical details when programming [4,5,7,8]. However, logical errors or abuse of permissions can also cause privilege escalation vulnerabilities and distory privacy protection. For example, the user's private data is illegally obtained by other users or even administrators. In this paper, a legal user accessing data by using permissions owed only by other users whose role is superior, inferior to or the same with him, can be all regarded as the category of privilege escalations.

In our previous research, five types of access control vulnerabilities based on the RBAC management model are proposed [9]. In this article, we further refine the permissions around the issue of privilege escalation based on hierarchical RBAC model and graph theory. Graph theory has many advantages in access control research [10, 11]. In this paper, the hierarchical relationship between roles in RBAC model is represented by a directed role graph, then the type of privilege escalations can be described clearly when an illegal request is accepted by an access control policy. There are three...
type privilege escalation, upward, downward and horizontal escalation, which are corresponding to the phenomena in the real world. In addition, another type of unauthorized access, that is, the illegal use of the permissions of other roles that are not related to the user’s role, is also studied in this paper. With the characteristics of access control vulnerabilities, it can be theoretically proved that whether an access control policy has illegal authorization and which type of vulnerabilities exists, which can be a theoretical basis for the actual access control policy vulnerability detection.

The structure of the entire article is as follows: Section 2 introduces the definition of the basic hierarchical RBAC model; Section 3 proposes multiple permissions concepts such as direct permissions, inherited permissions, and private permissions, and then analyses the properties of these permissions; Section 4 presents a method for determining access rights based on role graphs; Section 5 proposes optimization methods for general access control decisions; Section 6 summarizes the full text.

2. Basic Hierarchical RBAC Model

In many access control systems, users are divided into a certain number of categories according to their security levels or roles according to their capabilities and responsibilities and organized as a hierarchical structure. This hierarchy is produced due to some users having more access rights than others. To manage large systems, the hierarchical structure in RBAC is more complicated than other systems. The role hierarchy (RH) reflects the authority and responsibilities of an organization by constructing a hierarchical relationship between roles. This article makes appropriate modifications to the RBAC model [12] proposed by Sandhu et al and proposes a basic hierarchical RBAC model, defined as follows:

Definition 1 (Basic hierarchical RBAC model) The basic hierarchical RBAC model $\mathcal{M}=(\mathcal{U},\mathcal{R},\mathcal{P},\mathcal{UA},\mathcal{PA},\mathcal{RH})$, where $\mathcal{U}$, $\mathcal{R}$, $\mathcal{P}$ represent the set of users, roles and permissions, respectively, and $\mathcal{UA}\subseteq \mathcal{U}\times\mathcal{R}$, represents the set of relations between users and roles; $\mathcal{PA}\subseteq \mathcal{P}\times\mathcal{R}$, represents the set of relations between permissions and roles; $\mathcal{RH}\subseteq \mathcal{R}\times\mathcal{R}$ is a partial order relationship on $\mathcal{R}$, called role hierarchy or role dominated relation, and is denoted as $\geq$.

Note: Definition 1 ignores the concept session $\mathcal{S}$ and functions user and roles of the hierarchical RBAC model proposed by Sandhu. This article considers only the association between role hierarchy and permissions, so $\mathcal{PA}$ includes the relations between inherited permissions and roles, as well as directly assigning relations, while $\mathcal{UA}$ represents only the direct assignment of users to roles, which is the same as the RBAC model.

3. Permission Category Definition

The set of permissions possessed by a role $r\in \mathcal{R}$ is represented as $\mathcal{P}_r$. Role hierarchies considered here reflect the inheritance relationship of permissions. In order to further refine the permissions, this section introduces definitions about roles and users, respectively.

3.1. Permissions of Roles

This subsection mainly discusses the definition of permissions related to a role.

Definition 2 (Direct permissions of role $r$): Permissions that the system directly assigns to role $r$ are called direct permissions of role $r$, represented by $\mathcal{P}_r^d$.

Definition 3 (Inherited permissions of role $r$): For role $r'$ and $r$, satisfying $r'<r$, if $\forall p(p\in \mathcal{P}_r^d\land (p,r')\in \mathcal{PA})\rightarrow (p,r)\in \mathcal{PA}$, then $p$ is called a inherited permission of $r$ (that is, the permission inherited from the inferior role $r'$), which is represented as $\mathcal{P}_r^i$, and the set of inherited permissions of $r$ is represented by $\mathcal{P}_r^i$.

For the permission set $\mathcal{P}_r$ of role $r$, we have $\mathcal{P}_r=\mathcal{P}_r^d\cup \mathcal{P}_r^i$.

Property 1: $\mathcal{P}_r^d\land \mathcal{P}_r^i=\varnothing$.

Proof: According to the definition of $\mathcal{P}_r^i$, the proof is straightforward. Proved.

Definition 4 (Private permissions of role $r$): Private permissions of role $r$ are permissions that are directly assigned to $r$ and cannot be inherited by other roles, expressed as $\mathcal{P}_r^p$ such that $\mathcal{P}_r^p=\{p|(p,r)\in \mathcal{PA}\land \forall r'\in \mathcal{R}, p\in \mathcal{P}_r\land p\in \mathcal{P}_{r'}\rightarrow r'=r\}$.
Property 2: $P_r^s \subseteq P_r^d$.

**Proof:** Let $P_r$ be permission set of $r$ such that $P_r=P_r^d \cup P_r^i$. For $\forall p \in P_r^s$, according to Property 1: $P_r^s \cap P_r^i = \emptyset$, we have $p \in P_r^d$ or $p \in P_r^i$. If $p \in P_r^i$, then $\exists r' < r$ satisfies $p \in P_r'$, which contradicts $p \in P_r^s$, so only $p \in P_r^d$ holds. That is, for $\forall p \in P_r^s$, we have $p \in P_r^d$, therefore, $P_r^s \subseteq P_r^d$. **Proved.**

Property 3: If $p \in P_r^s$, for $\forall r' > r$, $p \in P_r'^s$, (i.e. private permissions cannot be inherited)

**Proof:** Suppose $p \in P_r^s$, then $\exists r' > r$ and $r'' = r$ such that $p \in P_r''$, (i.e. $p$ is inherited from $r''$), then we have $r'' = r$ (according to Definition 4), contradicting.

(2) $p$ is inherited from $r$. Given $p \in P_r^i$, $\exists r'' < r'$ and $r'' = r$ such that $p \in P_r''$, i.e. $p \in P_r$, then we have $P_r^s \subseteq P_r$, that is, all the permissions inherited by $r'$ are the permissions of $r$. Since $p \in P_r^i$, then $P_r^i \subseteq P_r$ implies $p \in P_r$ and $p \in P_r$. From $p \in P_r^s$, we know that $r = r''$ (Definition 4), which contradicts the premise $r' > r$. Therefore, for $\forall r' > r$, $p \in P_r^i$ holds. **Proved.**

### 3.2. Permissions of Users

This subsection mainly discusses the definition of permissions belong to a user.

**Definition 5:** (Direct permissions of user $u$): The permissions directly assigned to user $u$ by the system are direct permissions of $u$, expressed as $P_u^d$.

The set of inherited permissions of user $u$, i.e. permissions acquired by $u$ through the assigned role $r$, is represented as $P_u^i$. For role $r$, if $(u,r) \in UA$, then $P_u^i=P_r$. For $u$, the assigned role set $R_u=\{(u,r_i) \in UA, r_i \in \text{Role Set}\}$, so $P_u^i=\bigcup_{r_i \in R_u} P_r$. User $u$'s permission set $P_u=(\bigcup_{r_i \in R_u} P_r) \cup P_u^d$ (or written in the form of $P_u^d \cup P_u^i$).

**Corollary 1:** For user $u$ and role $r$, if $(u,r) \in UA$, then $P_r \subseteq P_u$.

**Proof:** It can be directly proved by $P_u=(\bigcup_{r_i \in R_u} P_r) \cup P_u^d$. **Proved.**

**Definition 6:** (Private permissions of user $u$): Private permissions of user $u$ are permissions that are directly assigned to $u$ and cannot be inherited by other users or any roles, denoted by $P_u^s$, $P_u^s=\{p|p \in P_u^d \land \forall u'u \in U, p \in P_u \rightarrow u' \neq u\}$.

In the RBAC system, the permissions obtained by users are generally through roles. Users’ special permissions or private permissions cannot be obtained through roles, so they can only be assigned directly. Then we have $P_u^s \subseteq P_u^d$. Permissions obtained through roles and permissions directly assigned to users should be treated differently.

**Theorem 1:** For role $r$ and user $u$, if $(u,r) \in UA$, then $P_r^i \cap P_u^s = \emptyset$.

**Proof:** Suppose $P_r^i \cap P_u^s \neq \emptyset$, i.e. $\exists p \in P_r^i \land p \in P_u^s$. If $p \in P_r^d$, we have $p \in P_u$, then $\exists u' \neq u$ satisfies $(u',r) \in UA$, and according to Corollary 1, we have $p \in P_u$. If $p \in P_u^s$, then $p \in P_u$ also holds. According to the definition of $P_u^s$, for $\forall u', p \in P_u \rightarrow u' = u'$, which contradicts the condition $u' \neq u$, therefore $P_r^i \cap P_u^s = \emptyset$. **Proved.**

**Corollary 2:** Given role $r$ and user $u$, if $(u,r) \in UA$, then $P_r^s \cap P_u^s = \emptyset$.

**Proof:** According to Theorem 1, $P_r^i \cap P_u^s = \emptyset$, according to Property 2: $P_r^s \subseteq P_r^d$, then $P_r^s \cap P_u^s = \emptyset$. **Proved.**
In order to analyse several phenomena of unauthorized access, this paper firstly analyse the basic hierarchical RBAC model and describes the role hierarchy relationship in form of role graph.

4.1. Directed Role Graph
This subsection mainly discusses the relation between permissions and roles through a directed role graph.

Definition 6 (Direct role relationship): For two roles \( r_i \) and \( r_j \), satisfying \( r_i \rightarrow r_j \), if there is no role \( r_m \), such that \( r_m \rightarrow r_i \) and \( r_m \leftarrow r_j \), then \( r_i \) and \( r_j \) form a direct role relationship, denoted by \( r_i \rightarrow r_j \), where \( r_i \) is the direct superior role of \( r_j \) and \( r_j \) is the direct inferior role of \( r_i \).

There may be more one roles direct superior or inferior to role \( r \).

Definition 7 (Direct inferior role function): Let \( dh: R \rightarrow 2^R \), and function \( dh(r) \) represents the set of all direct inferior roles of \( r \), where \( dh(r)=\{r' \mid r \rightarrow r', r \in R\} \).

Definition 8 (Direct superior role function): Let \( dh^{-1}: R \rightarrow 2^R \), and function \( dh^{-1}(r) \) represents the set of all direct superior roles of \( r \), where \( dh^{-1}(r)=\{r' \mid r \leftarrow r', r \in R\} \).

Definition 9 (Role graph): Given a basic hierarchical RBAC model \( M \), the role graph is a directed acyclic graph \( G_M=(R,E) \), where \( R \) is the set of roles and also the set of \( G_M \) nodes, for each \( r \in R \), \( r \) is a node of \( G_M \), and \( E \) is a set of directed edges \( e=(r_i, r_j) \), where \( E=\{(r_i, r_j) \mid r_i \rightarrow r_j, r_i, r_j \in R, i,j \in \{0, ..., |R|-1\}, i \neq j\} \).

For a basic hierarchical RBAC model \( M \), the relations between roles can be described by more than one sub-directed role graphs, such as \( G_{M1}, G_{M2}, ..., G_{Mn} \). Thus, given role \( r \in R \) and \( r' \in R \), if:

1) \( r \) and \( r' \) are roles in the same role graph, and 1) If \( r \) is the ancestor of \( r' \) (including parent nodes), denoted by \( r \rightarrow r' \), then \( r \) can inherit \( r' \)’s permissions; 2) If \( r \) is a descendant of \( r' \) (including child nodes), denoted by \( r \leftarrow r' \), then the non-private permissions of \( r \) can be inherited by \( r' \); 3) If \( r \) and \( r' \) are the same role (i.e., the same node), denoted by \( r=r' \), then the set of permissions of \( r \) and \( r' \) are equal; 4) If \( r \) is neither an ancestor nor a descendant of \( r' \) and \( r \not\rightarrow r' \), then we say \( r \) and \( r' \) are brothers, represented by \( r \not\rightarrow r' \) and the permissions of \( r \) and \( r' \) can be inherited by their common ancestors, but \( r \) and \( r' \) are not related to each other.

2) If \( r \) and \( r' \) are in different role graphs, then \( r \) and \( r' \) are not related, denoted by \( r \not\rightarrow \not\rightarrow r' \).

Figure 1 shows a complex example of \( G_M \), where \( R=\{r_1, r_2, ..., r_{18}\} \). \( G_M \) contains two sub-role graphs. \( r_1 \) is the parent node of \( r_7 \) and \( r_6 \), and the ancestor of \( r_{10} \) and \( r_{11} \); \( r_{12} \) and \( r_{13} \) are descendants of \( r_7 \); \( r_6 \) and \( r_7 \) are unrelated (brother), and so on.

![Figure 1. Example of role graphs](image)

We assume that all roles have one super role (denoted by \( r_{super} \)), thus we can merge all sub graphs into one role graph. For example, the super administrator can merge the above role graphs into one, including a start node and multiple end nodes. In this way, those nodes that are not related to \( r \) can be considered as brother nodes of \( r \), here we use the notation \( r'r \) to represent both \( r \not\rightarrow r' \) and \( r \not\rightarrow \not\rightarrow r' \). As shown in Figure 2, the unrelated relationships are \( r_7 \not\rightarrow r_2, r_7 \not\rightarrow r_6, r_7 \not\rightarrow r_8 \), etc.
Definition 10 (Path): Given a basic hierarchical RBAC model $M$, the alternately appearing sequence of nodes and edges in the role graph $G_M=(R,E)$ is denoted as $\gamma$, such that $\gamma=r_0e_1r_1e_2...e_kr_{k-1}$, if two ends of edge $e_i$ in $\gamma$ are $r_{i-1}$ and $r_i$ such that $r_{i\rightarrow i+1}$ ($i\in\{0,...,|R|-1\}$), then $\gamma$ is called a path from $r_0$ to $r_k$.

Let $R_i$ be the set of nodes such that $R_i=\{r_0, r_1,..., r_k\}$, where $r_i (i=0...k)$ is the node contained in $\gamma$ and $R_{\gamma}$ be the set of nodes in $\gamma$, where $\gamma$ denote the path that pass through role $r$. Obviously, $r\in R_{\gamma}$. Then we use $\Gamma_r$ to represent the set of all paths passing through $r$, such that $\gamma_r\in \Gamma_r$. Let $R_{\gamma}$ denote the set of nodes in $\Gamma_r$, where $R_{\gamma}\subseteq R$, and $\overline{R_{\gamma}}$ represents the set of nodes that are not in $R_{\gamma}$, such that $\overline{R_{\gamma}}=R\setminus R_{\gamma}$. For $r'\in R$, if $r'\in R_{\gamma}$, then $r'$ is called the related node of $r_i$, if $r'\not\in R_{\gamma}$, then $r'$ is called the node unrelated with $r_i$. For $r\in R_{\gamma}$, $\Gamma_r$ is represented as $\gamma_i$. For $r'\in R_{\gamma}$, $\Gamma_r$ is represented as $\gamma_{i\rightarrow i+1}$.

4.2. Access Request and Access Control Strategy
In order to analyse the possible privilege escalation in the access control policies, we assume that there is an illegal access control request that is accepted by the access control policy, which means there is a logic error in the access control policy. Then, we give the following definitions.

Definition 11 (Access request): An access request is that a user $u$ requests to access the resource $o$ by an action $a$, expressed by $\text{Req}(u,a,o)$.

Definition 12 (Access control rules and access control policies): Let $\rho=(u,a,o,c)$ denote an access control rule, where $u$, $a$, $o$, and $c$ represent a user, action, resource and constraint, respectively. The set of access control rules is called an access control policy, denoted by $\pi$.

Since our discussion is based on Sandhu's RBAC96 model, here we ignore the details of resources and actions, and $(a,o)$ is directly represented as $p$. Thus the access request is expressed as $\text{Req}(u,p)$, and the access control rule is represented as $\rho=(u,p,c)$.

Given an access control policy $\pi$ and access request $\text{Req}(u,p)$, we use $\pi\vdash\text{Req}(u,p)$ to indicate $\pi$ accept the access request, so the access control system grants $u$ permission $p$; and $\pi\not\vdash\text{Req}(u,p)$ to indicate $\pi$ denies the access request, so the system refuses to grant $p$ to $u$.

As a representation of the hierarchical relationship of roles, role graph can also reflect the inheritance relationship of permissions. Take Figure 3 as an example. The bold directed edges form all the paths of $\Gamma_r$ passing through node $r$, and bold nodes are the nodes on $\Gamma_r$. Let $R_{\Gamma_r}$ represent all nodes that related with $r$. For roles on $\Gamma_r$, such as the role $r\rightarrow r_i$ will inherit $r_i$ permissions and the role $r\leftarrow r_i$ will be inherited by $r_i$, and users with the same role as $r_i$ will have the same permission set as $r_i$. Therefore, all the nodes on $\Gamma_r$ are associated with $r_i$. For an access request $\text{Req}(u,p)$, if $(u,r)\in UA$, $p\in P_r$, where $r\in\Gamma_r\setminus\{r_i\}$ and $p\not\in P_{r_i}$, then if $\pi\vdash\text{Req}(u,p)$, it indicates that $\pi$ has a logic error.
According to the previous analysis and Figure 3, the illegally acquired permissions can only be in two sets, one set is related to the elements in $R_{r_7}$ (except $r_7$), and the other is related to the elements in $\bar{R}_{r_7}$. Therefore, the escalation of $r_7$ can be divided into two types: related to $r_7$ and unrelated to $r_7$.

4.3. Privilege Escalation Analysis

There are three types of related escalations: upward escalation, downward escalation, and horizontal escalation. Figure 3 shows that the $r_7$ node is the reference point where the upward, downward, and horizontal unauthorized nodes refer. To simplify the description, we denote the three kinds of vulnerabilities as Up-Privilege-Escalation (UPE), Down-Privilege-Escalation (DPE) and Horizontal-Privilege-Escalation (HPE).

**Definition 13** (Related privilege escalation vulnerabilities): For an access control policy $\pi$ and an access request $\text{Req}(u,p)$, if $(u,r) \in UA$, then $\text{Req}(u,p)$ is an illegal request, and if $\exists r'$ satisfies $p \in P_{r'}$, then (1) If $r' > r$, then $\text{ Req}(u,p)$ is upward unauthorized access, and if $\pi\text{-Req}(u,p)$, then $\pi$ contains UPE vulnerabilities, expressed by $\pi\in\text{UPE}$; (2) If $r' < r$, then $\text{Req}(u,p)$ is downward unauthorized access, and if $\pi\text{-Req}(u,p)$, then $\pi$ contains DPE vulnerabilities, represented by $\pi\in\text{DPE}$; (3) If $r' = r$, then $\text{Req}(u,p)$ is horizontal unauthorized access, and if $\pi\text{-Req}(u,p)$, then $\pi$ contains HPE vulnerabilities, expressed by $\pi\in\text{HPE}$.

**Theorem 2**: For a role $r \in R$, $\exists r > r$, $P_{r'} \cap P_r = \emptyset$. 
**Proof**: Suppose $P_{r'} \cap P_r \neq \emptyset$. For $r > r$, then $\exists \pi \in P_{r'}$ and $P \in P_r$. With respect to Property 1: $P_{r'} \cap P_r = \emptyset$, there are two cases discussed as following:

1. If $p$ is not the private permission of $r$, then $p$ can be inherited by $r'$, thus $p \in P_{r'}$, according to the assumption $p \in P_{r'}$, we know $P_{r'} \cap P_r = \emptyset$, which contradicts Property 1;

2. If $p$ is the private permission of $r$, i.e., $p \in P_{r'}$, implies $p \in P_r$. According to the assumption, $\exists \pi \in P_{r'}$, thus $p \in P_r$, then according to the definition of $P_{r'}$, there must be $r' = r$, contradicting.

Therefore, $\exists r > r$, $P_{r'} \cap P_r = \emptyset$ holds. **Proved**.

**Theorem 3**: Given an access control policy $\pi$ and an access request $\text{Req}(u,p)$ such that $(u,r) \in UA$, if $\pi\text{-Req}(u,p)$, then (1) If $\exists r > r$, $p \in P_{r'}$, then $\pi\in\text{UPE}$; (2) If $\exists r < r$, $p \in P_{r'}$, then $\pi\in\text{DPE}$; (3) If $r' = r$, $\exists u \neq u$, such that $(u',r') \in UA$ and $p \in P_{u}$, then $\pi\in\text{HPE}$.

**Proof**: (1) According to Theorem 2: $P_{r'} \cap P_r = \emptyset$, we know that for $\exists r > r$, if $p \in P_{r'}$, then $p \notin P_r$. Therefore, for $r > r$, $\pi\in\text{UPE}$.

(2) For role $r$, $P \cap P_{r'}$, suppose $p \in P_r$. According to Property 1, we know $p \in P_{r'}$ or $P \in P_{r'}$. If $P \in P_{r'}$, from Property 3, we have $P \in P_{r'}$, then we only need to consider the situation of $p \in P_{r'}$, $p \in P_{r'}$ implies $p \in P_r$, thus $P_{r'} \cap P_r \neq \emptyset$. According to Theorem 2, for $r > r'$, we have $P_{r'} \cap P_r = \emptyset$, which is contradictory, so the assumption is not hold, we have $p \notin P_r$. Therefore, for $r' < r$, $\pi\in\text{DPE}$.

(3) For $r' = r$, $\exists u \neq u$, if $(u',r') \in UA$ and $p \in P_{u}$, then $p \in P_{u}$, according to Corollary 3, if $p \in P_r$, then $p = P$, then according to the definition of $P_{r'}$, $u \neq u$ is impossible. Thus, $p \notin P_r$. Therefore, for $r' = r$, $\pi\in\text{HPE}$. **Proved**.

Theorem 3 can be used to determine whether there is an $r$-related unauthorized loophole and its type. However, how to determine the authorization of illegal access that does not satisfy the condition of Theorem 3? The following analysis is still based on Theorem 3.

For the (1) of Theorem 3, $\exists r > r$, we further discuss the case that $p \in P_r$. From $P \cap P_{r'}$, and Property 1, we know $p \in P_{r'}$ or $P \in P_{r'}$. We have proved that $p \in P_{r'}$, which implies $\pi\in\text{UPE}$, thus here we only consider if $p \in P_{r'}$, $\pi\in\text{UPE}$?

If $p \in P_{r'}$ and $r > r$, then the following two inheritance case are discussed:

1. If $p$ is not inherited from $r$, because $p \in P_{r'}$ excludes the possibility of $p \in P_{r'}$, thus if $p \in P_{r'} \setminus P_{r'}$, it must be the case that $r'$ inherits $p$ from $r$, which is contrary to the premise, therefore, $p \notin P_r$. Then $\exists r''$ satisfies $r'' > r'$ and $r'' > r$, such that $r'$ inherits $p$ from $r''$, then we have $p \in P_{r'}$, and

2. If $r'' \in R_{r'}$, then if $r'' > r'$, assume $r'' = r'$, however this case should be excluded, otherwise the discussion is meaningless, thus, $r'' > r'$; therefore we consider only the case of $r'' < r$. If $p \in P_{r''}$, then $p$ cannot be inherited, which contradicts the premise of $p \in P_{r'}$, so $p \notin P_{r'}$; Then from $p \notin P_{r'}$, $r'' < r$ and
If \( p \in P_r \), we know that \( r \) can inherit \( p \) from \( r'' \), i.e., \( p \in P_r \), which is inconsistent with the premise of 1); therefore \( r'' \in R_{r''} \).

2) If \( r'' \in R_{r''} \), that is \( r'' \) is an unrelated node with \( r \), obviously \( r' \) is the common ancestor of \( r'' \) and \( r \), if \( p \) is not the permission inherited from \( r \), then according to the previous inference, \( p \) must be inherited from \( r'' \).

Therefore, for \( p \in P_r \), if \( p \) is not inherited from \( r \), then there must be \( p \notin P_r \) and \( p \notin P_r \); where \( r'' \in R_{r''} \). Therefore, \( \text{Req}(u, p) \) is an illegal request, or \( \pi \text{–Req}(u, p) \) has an error that allows illegal access. Since \( r'' > r \), thus \( \pi \notin \text{UPE} \).

(2) If \( p \) is inherited from \( r \): we have \( p \in P_r \), which means that \( \text{Req}(u, p) \) is a legal request, and \( \pi \text{–Req}(u, p) \) means that \( \pi \) accepts a legal access.

From the above discussion, we conclude that for \( r'' > r \), only when \( p \in P_r \), \( \pi \in \text{UPE} \).

UPE, DPE and HPE vulnerabilities can be called \( r \)-related vulnerabilities, and Theorem 3 can be taken as \( r \)-related vulnerability determination theorem. The access control vulnerabilities that do not satisfy Theorem 3 is called \( r \)-unrelated vulnerabilities. We further analyze the \( r \)-unrelated vulnerability and give its definition as followings.

**Definition 14** (Unrelated privilege escalation vulnerability): Given an access control policy \( \pi \) and an access request \( \text{Req}(u, p) \), if \((u, r) \in \text{UA} \) and \( p \in P_r \), then \( \text{Req}(u, p) \) is an illegal access, if \( \exists r'' \) such that \( p \in P_r \) and \( r'' > r \), then \( \pi \) has an \( r \)-unrelated-Privilege-Escalation vulnerability (abbreviated as UPER).

For simplification, for \( r \) satisfying \( r'' > r \), the set of permissions inherited from \( r \) in \( P_r \) is denoted by \((P_r)^i \), and the set of those not inherited from \( r \) is \((P_r)^f \), such that \((P_r)^i \cap (P_r)^f = \emptyset \), we have \( P_r = (P_r)^i \cup (P_r)^f \) and obviously, \((P_r)^i \) \(\subseteq P_r \).

For an access request \( \text{Req}(u, p) \), such as \((u, r) \in \text{UA} \), if \( p \notin P_r \), then \( p \notin P_r \); \((r'' > r) \) (except \( P_r^d \) otherwise it belongs to the UPE problem), then \( p \) must be inherited from the node that is not associated with \( r \). Thus, we give Theorem 4 and the corresponding proof. For the convenience of expression, we use the symbol \( \mathbb{P}_r^d \) to denote the set of directly assigned permissions for all the roles \( \forall r \) that \( > r \) in \( R_{\Gamma_r} \).

**Theorem 4**: Given the basic hierarchical RBAC model \( M \) and its role graph \( G_M = (R, E) \), for \( r \) and \( r'' \in R \) satisfying \( r'' > r \), if \( \exists p \in \mathbb{P}_r^d, p \in P_r \) and \( p \notin P_r \), then \( p \in (P_r)^i \), where \( r'' > r \) and \( r'' \in R_{\Gamma_r} \).

**Proof**: For \( r'' > r \), if \( p \in P_r \), then

1) Prove that the direct advanced role \( r \) of \( r \) (i.e. \( r, \rightarrow r \)), if \( \exists \exists p \in \mathbb{P}_r^d, p \in P_r \) and \( p \notin P_r \), then \( p \in (P_r)^i \), where \( r'' \) satisfies \( r, \rightarrow r'' \) and \( r'' \in R_{\Gamma_r} \). The proof is as follows:

From \( P_r = \mathbb{P}_r^d \cup P_r \) and Property 1, we have \( \exists p \in P_r \) satisfies \( p \in \mathbb{P}_r^d \) or \( p \in P_r \). According to the condition \( p \in \mathbb{P}_r^d \), we have \( p \notin P_r \), so only \( p \in P_r \) holds. From \( r, \rightarrow r \), we have \( p \in (P_r)^i \) or \( p \notin (P_r)^i \). If \( p \in (P_r)^i \), then \( p \in P_r \), which contradict the condition \( p \in P_r \), therefore, \( p \notin (P_r)^i \), then \( p \in (P_r)^f \), which means that \( r'' \) cannot inherit the permission \( p \) from \( r \), and can only inherit \( p \) from other direct low-level roles. Then \( \exists r'' \) \( \rightarrow r'' \) such that \( r, \rightarrow r'' \), \( p \in (P_r)^i \), if \( r'' \rightarrow R_{\Gamma_r} \), there are two cases to consider:

- \( r'' > r \) or \( r'' < r \).

1) If \( r'' < r \), according to \( p \in (P_r)^i \), we know that \( p \in P_r \) and \( p \notin P_r \), then \( p \notin P_r \), which contradicts the condition \( p \in P_r \). Hence, \( r'' < r \) is not hold.

2) If \( r'' > r \), from \( p \in (P_r)^i \), we have \( p \in P_r \) and according to the condition \( p \notin \mathbb{P}_r^d \), we have \( p \notin \mathbb{P}_r^d \), so \( p \in (P_r)^i \). Given \( r'' > r \), there is \( p \in (P_r)^i \) which contradicts the previous inference, we know that \( p \notin (P_r)^i \), then only \( p \in (P_r)^f \), therefore it must be \( \exists r'' \not\rightarrow r \) and \( r'' \not\rightarrow r \), such that \( p \in (P_r)^i \). At this step, we still need to discuss the situation of \( r \in R_{\Gamma_r} \) where \( r < r \) (\( r < r \) has been proved to be invalid). As long as the condition \( r'' \rightarrow r \) is satisfied, we can find an infinite number of nodes, however, \( R_{\Gamma_r} \) is a finite set, therefore, for \( r'' \rightarrow R_{\Gamma_r} \), \( r'' > r \) is not hold.
According to the inference above, we have \( r'' \notin R_{\Gamma_r} \). From \( r \mapsto r'' \) and \( r \mapsto r \), we conclude \( r'' \notin R_{\Gamma_r} \).

(2) For \( \forall r \mapsto r \) on the set \( R_{\Gamma_r} \), if \( \exists p_r \in \mathbb{P}_r^d, p_r \in P_r \) and \( p_r \in P_r \), we need prove that \( p_r \in (P_r)^1 \), where \( r \mapsto r'' \) and \( r'' \notin R_{\Gamma_r} \).

From \( P_{rz} = P_{r_1}^1 \cup \mathbb{P}_r^d \), we know that \( p_r \in P_{r_1}^1 \), for \( \forall r \) satisfies \( r \mapsto dh(r_1) \), we have \( P_{r_1}^1 = \cup_{r \mapsto dh(r_1)} (P_r \mathbb{P}_r^d) \), then \( dh(r_1) \) can be further decomposed into two sets, \( \Phi_1 \) and \( \Phi_2 \), where \( \Phi_1 = (dh(r_1) \cap R_{\Gamma_r}) \) and \( \Phi_2 = (dh(r_1) \cap \mathbb{R}_{\Gamma_r}) \), obviously \( \Phi_1 \cap \Phi_2 = \emptyset \), which means that the permissions in \( P_{r_1}^1 \) can be inherited from the role node in \( \Phi_1 \) or \( \Phi_2 \). Obviously, \( \Phi_1 \) is the set of \( r \)-related nodes, and \( \Phi_2 \) is the set of \( r \)-unrelated nodes.

Assuming that the permissions in \( P_{rz}^1 \) can be inherited from the node in \( \Phi_1 \), that is, for \( r_m \in \Phi_1 \), if \( p_r \in P_{r_m}^1 \), we have \( p_r \in (P_{r_1}^1) \), then we need to consider three cases: \( r_m \mapsto r, r_m \mapsto r \), and \( r_m \mapsto r \).

1) For the case of \( r_m \mapsto r \). i.e., \( r \mapsto r_m \), according to the analysis of (1), we know that this situation cannot happen.

2) For the case of \( r_m \mapsto r \). As we have known that \( r \mapsto r \) and \( r \mapsto r_m \), so it is contradictory, thus, this situation cannot happen.

3) For the case of \( r \mapsto r \). If \( p_r \in P_{r_m}^1 \), according to \( P_{r_m}^1 = P_{r_1}^1 \cup \mathbb{P}_r^d \), and \( p_r \in \mathbb{P}_r^d \), we know that \( p_r \in P_{r_1}^1 \), according to the previous steps for \( P_{r_1}^1 \), we can further analyze the direct low-level roles of \( r_m \), and according to the previous analysis of (1) and (2), we can decompose roles that \( \mapsto r \) and analyze them one by one, \( r_m \mapsto r \mapsto r_m \mapsto r \mapsto r \), until the end \( r \mapsto r \), that is, \( r_m \) is the direct senior role of \( r \). According to the analysis of (1), we finally find the satisfying node \( e \notin R_{\Gamma_r} \), so this case cannot happen.

In summary, the permissions in \( P_{r_1}^1 \) can only be inherited from the node in \( \Phi_2 \), that is, \( \exists r_m \in \mathbb{R}_{\Gamma_r} \), which satisfies \( p_r \in (P_{r_m}^1) \).

Since \( r \) is a node arbitrarily selected from \( R_{\Gamma_r} \) that satisfies \( r \mapsto r \), the theorem is proved according to the analysis of (1) and (2). **Proved.**

Theorem 3 directly determines an access request based on the characteristics of related vulnerabilities, and at the same time judges exactly which vulnerabilities it belongs to. However, when none of the access requests meets the characteristics of associated vulnerabilities, we can only determine whether it is a legitimate access, and then determine whether it is an unassociated vulnerability according to Theorem 4.

**Corollary 4** Given an access control strategy \( \pi \) and an access request \( \text{Req}(u, p) \) such that \( (u, r) \in UA \), if \( p \in P_r \), and \( p \in \mathbb{P}_r^d \), and \( \exists r \mapsto r \) such that \( p \in P_r \), then if \( \pi \mapsto \text{Req}(u, p) \), \( \pi \) contains an unrelated privilege escalation vulnerability.

**Proof:** According to Theorem 4, \( \exists r'' \) such that \( p \in (P_r)^1 \), \( r \mapsto r'' \) and \( r'' \notin R_{\Gamma_r} \), That is, the relation between \( r'' \) and \( r \) is \( r \mapsto r'' \). According to Definition 14, \( \pi \) contains an unrelated privilege escalation vulnerability. **Proved.**

5. Access Control Decision Optimization Algorithm and Comparison

Making decision about an access request is a complicated process, especially in a complex application scenario, where the excessive number of roles can easily make it difficult to query permissions. In this section we discuss how to use previous theories to improve the efficiency of access request decisions.

5.1. Calculating Formulas of Permissions

To calculate the permissions, we first give the definition of set of users that directly assigned to roles.

**Definition 15** (User function): User function \( u: R \rightarrow 2^U \), \( u(r) = \{ u | (u, r) \in UA, u \in U \} \).

Given a basic hierarchical RBAC model \( M \) and its directed role graph \( G_M=(R, E) \). For an access request \( \text{Req}(u, p) \), where \( (u, r) \in UA \), how to find \( p \in P_r \)? There are two ways to consider: (1) Directly searching for the existence of \( p \) in \( P_r \); or (2) Searching among various related nodes.
We use formulas in Table 1 to compute each type of permissions, where inherited permissions are directly expressed in a recursive form. For convenience of expression, the union (∪) and subtraction (\(\setminus\)) symbols of the set are replaced by the addition (+) and subtraction (-) in arithmetic operations. Table 1 shows all the permission expressions.

In Table 1, Formula (1) is already included in Formula (2) and Formula (6) is the calculation of unrelated permissions, which can be ignored. Therefore, we need only consider (2) ~ (5). To further simplify (2) ~ (5), we use simple set operation instead of recursive calculation by directly accumulating the permissions possessed by all role nodes and subtracting the private permissions of all roles. So each non-recursive formula in the last column has new number (2') ~ (5') corresponding to their recursive formula on the second column.

### Table 1. Formula of permissions

| Formula Number | Recursive formula | New number | Non-recursive formula |
|----------------|-------------------|------------|----------------------|
| (1)            | \( P^d_r = \bigcup_{r' \in dh(r)} (P^d_{r'} \cup (P^d_r \setminus P^d_{r'})) \), \( dh(r) \neq \emptyset \) | - | - |
| (2)            | \( P_r = P_r^d \cup \bigcup_{r' \in dh(r)} (P_r \setminus P^d_{r'}) \), \( dh(r) \neq \emptyset \) | (2') \( P_r = P_r^d + \sum_{i=1}^{v} P^d_{r_i} - \sum_{i=1}^{v} P^d_{r_i'} \) | \( r_i < r \) and \( y = \{r_i | r_i < r \} \) |
| (3)            | \( P_r^d = \bigcup_{r' \in dh(r)} P^d_{r'} \), \( |dh(r)| \geq 2 \) | (3') \( P_r^d = \sum_{i=1}^{v} P^d_{r_i} \) | \( r_i > r \) and \( y = \{r_i | r_i > r \} \) |
| (4)            | \( P_u^r = \bigcup_{r' \in u(a)} P^r_{u'} \), \( |dh(r)| = 1 \land r' \in dh(r) \) | (4') \( P_u^r = \sum_{i=1}^{x} P^r_{u_i} \) | \( r_i < r \) and \( x = \{r_i | r_i < r \} \) |
| (5)            | \( P_u^r = \bigcup_{u \in u(a)} P^r_u \), \( |ua(r)| \geq 2 \land u' \neq u \) | (5') \( P_u^r = \sum_{i=1}^{z} P^r_{u_i} \) | \( (u, r) \in UA \) and \( z = \{u_i | (u, r) \in UA \} \) |
| (6)            | \( P_r = \bigcup_{r' \in R_r} P^r_{r'} \) | - | - |

#### 5.2. Optimization of Access Control Decision Algorithm and Comparison

For an access request Req \((u, p)\) such that \((u, r) \in UA\), the general decision-making method is to search \(p\) in the \(P_r\) set. From formula (2') we know that in the worst case, all inferior roles need to be searched (the number is \(x\)), which include direct and private permissions.

Given \(G_M\) and \(P_r^d, P_r^r\) of each role and \(P_u^r\) of \(r\)’s other users, a non-optimized decision algorithm is expressed as follows:

Step 1: search \(p\) in \(P_r^d(r' < r)\) and \(P_r^r\), and determine whether \(p\) is a private permission, if \(p \in (P_r^d \cup P_r^r)\), then: 1) if \(p \notin P_u^r\), return "Legal", and exit the algorithm; 2) Otherwise, return "Illegal, DPE", and exit; otherwise, if \(p \notin (P_r^d \cup P_r^r)\), go to Step 2.

Step 2: Search \(p\) in \(P_u^r\), if \(p \in P_u^r\), then return "Illegal, UPE", and exit; otherwise, go to Step 3.

Step 3: Search \(p\) in \(P_u^r\), if \(p \in P_u^r\), returns "Illegal, HPE" and exit; otherwise, returns "Illegal, URPE" and exit the algorithm.

In order to optimize the algorithm and compare the result before and after optimization, we use the number of roles searched in the worst case to represent the time consuming in calculating permissions of each role scanned, and the time that scan one direct permission is represented as a unit 1. Then we use the following symbols to represent the corresponding process: 1) \(z'\): the time to scan the privacy permissions of all users of \(r\); 2) \(x'\): the time to scan the privacy permissions of all roles inferior to \(r\); 3)
\( y^d \): time to scan the direct permissions of all roles superior to \( r \); 4) \( x^d \): time to scan all of direct permissions of roles inferior to \( r \); 5) \( z^d \): time to scan all direct permissions of roles inferior to \( r \), excluding the privacy permissions; 6) \( 1^d \): time to scan \( r \)'s direct permissions, which equals to unit 1.

The flow chart of non-optimized algorithm is shown in Figure 4, where the number (1)-(4) refer to each final decision-making process. The most time-consuming is to directly scan the permissions of the roles superior and inferior to \( r \). In order to optimize the algorithm, we consider three types of vulnerabilities mentioned in Section 4, try to minimize the number of scanned roles superior and inferior to \( r \). All procedures involving \( x \) and \( y \) should be placed behind other procedures as much as possible. Thus, we can optimize the decision algorithm as follows:

1. **Step 1:** Scan \( P_u^d \). If \( p \in P_u^d \), return "Illegal, HPE" and exit the algorithm. Otherwise, go to Step 2;
2. **Step 2:** Scan \( P_r^d \). If \( p \in P_r^d \), return "Illegal, DPE" and exit the algorithm. Otherwise, go to Step 3;
3. **Step 3:** Scan \( P_r^d \). If \( p \in P_r^d \), return "Illegal, UPE" and exit the algorithm. Otherwise, go to Step 4;
4. **Step 4:** Scan \( P \). If \( p \in P \), return "Legal" and exit the algorithm. Otherwise, return "Illegal, URPE" and exit the algorithm.

The flow chart of optimized algorithm is shown in Figure 5.

**Figure 4.** Flow chart of non-optimized access decision algorithm

**Figure 5.** Flow chart of the optimized decision algorithm

To compare the time consumption of algorithms before and after optimization, we still use the number of roles scanned in the worst-case to calculate the complexity of each procedure. Table 2 shows the result of every process.

**Table 2.** Time consumption before and after optimization

| Results         | Before optimization          | After Optimization       |
|-----------------|-----------------------------|--------------------------|
| Legal           | \((1)1^d+x^d+x^d\)           | \((4)z^d+x^d+y^d+1^d+x^d\) |
| DPE             | Simultaneously with (1)      | (2) \(z^d+1^d\)           |
| UPE             | \((2)1^d+x^d+x^d+y^d\)       | \((3)z^d+x^d+y^d\)         |
| HPE             | \((3)1^d+x^d+x^d+y^d+z^d\)   | (1) \(z^d\)               |
| URPE            | \((4)1^d+x^d+x^d+y^d+z^d\)   | \((5)z^d+x^d+y^d+1^d+x^d\) |
| Total           | \(4\times1^d+4\times x^d+4\times x^d+3\times y^d+2\times z^d\) \((*)\) | \(5\times z^d+4\times x^d+3\times y^d+2\times x^d+2\times 1^d+2\times x^d\) \((**)\) |

To compare, \((*)-(**)\) = \(2\times x^d-3\times z^d\), which shows the efficiency of the optimization depends on the number of \( x \) and \( z \), eventually in the worst case. However, the optimization method is not unique, we can further discuss it around the number of roles and their hierarchy according to the actual environment.

Table 2 also shows that if we need to know whether an access request is legal, we can use the process before optimization (1). If we only need to know whether there is DPE, UPE or HPE, we can directly use one of the optimized steps (1)(2)(3). For some application systems, such as applications with a relatively small number of advanced roles, we can quickly detect the presence of UPE.
6. Conclusion

According to permission inheritance in the role hierarchy, the concept of permission is divided into three categories: direct permissions $P_d$, inherited permissions $P_i$ and private permissions $P_p$. On this basis, role hierarchical relations are represented by a directed role graph in which the possible decision results of an access request and access control strategy are studied. According to the inheritance relationship between the roles associated with role $r$ in the role graph and permissions, this paper proposes a determination theorem about UPE, DPE and HPE, and concludes that: if $p \in P^D_H$, the policy contains HPE; if $p \in P^U_H$, the policy contains DPE; and if $p \in P^U_I$, the policy contains UPE. In addition, it is also proved that if a request $\text{Req}(u, p)$ is an illegal access and it does not belong to any of above three vulnerabilities, the policy contains URPE vulnerabilities. Based on the above conclusions, this paper finally gives the optimization method of access control decision.

We will optimize the access control decision algorithm in various application scenarios and evaluate the efficiency of the optimization as our future work.

7. Acknowledgement

This work is supported by Science and technology research project of Jiangxi Provincial Department of Education (170678) and National Natural Science Foundation of China (61861020, 61762044).

8. References

[1] Fang L, Yin L H, Guo Y C. A Survey of Key Technologies in Attribute-based Access Control Scheme. Chinese J. of Computers, 2017, 40(7):1680-98 (in Chinese)

[2] OWASP. Owasp top 10- 2013 the ten most critical Web application security risks. [2013-06-12]. https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013.

[3] OWASP. OWASP Top 10 Application Security Risks - 2017. https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_Top_10.html

[4] Zhu J, Chu B, Lipford H, et al. Mitigating Access Control Vulnerabilities through Interactive Static Analysis Acm Symposium. ACM, 2015 p 199-209

[5] Mulliner C, Robertson W, Kirda E. Hidden GEMs: Automated Discovery of Access Control Vulnerabilities in Graphical User Interfaces. 2014 IEEE Symposium on Security and Privacy p 149-62

[6] Monshizadeh M, Naldurg P, Venkatakrishnan VN. Mace: Detecting privilege escalation vulnerabilities in web applications. The 2014 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2014. p 690-701.

[7] Zhu J, Chu B, Lipford H. Detecting Privilege Escalation Attacks through Instrumenting Web Application Source Code. Proc. of ACM on Symposium on Access Control Models & Technologies, 2016 p 73-80.

[8] Wen S, Xu J, Yuan L Y, Xu S H and Si G N. A Test Case Generation Approach for Exploiting Access Control Vulnerabilities Based on Policy Inference. Chinese Journal of Computers, 2017, 40(12):2658-70 (in Chinese)

[9] Ma L, Yan Y, Xie H. A new approach for detecting access control vulnerabilities. Proc. of the 7th Int. Conf. on Information, Communication and Networks (ICICN 2019), Institute of Electrical and Electronics Engineers Inc., 2019 p 109-13

[10] Alvesa S, Fernandesb M. A graph-based framework for the analysis of access control policies. Theoretical Computer Science. 2017, 685 p 3-22

[11] Koch M, Mancini L V, et al. A graph-based formalism for RBAC. ACM Transactions on Information and System Security, 2002, 5(3): 332–65

[12] Sandhu R, Coyne E, Fenster H, Yourman C. Role-based access control models. IEEE Computer, 1996, 29(2): 38-47