

# On Permissions, Inheritance and Role Hierarchies

Jason Crampton

Information Security Group  
Royal Holloway, University of London

# Introduction

- The role hierarchy is central to most RBAC models
  - Modelled as a partially ordered set  $R$
- The hierarchy makes use of two types of inheritance
  - If permission  $p$  is assigned to role  $r$  then  $p$  is implicitly assigned to all roles  $s$  such that  $r < s$
  - If user  $u$  is assigned to role  $r$  then  $u$  is implicitly assigned to all roles  $s$  such that  $s < r$

# Introduction – Motivation

- Consequences of role hierarchy and inheritance
  - ✓ Simplification of administration through reduction in number of permission-role and user-role assignments
  - ✓ Natural mapping of enterprise characteristics (organizational hierarchy) onto access control model (role hierarchy)
  - ✗ Senior roles have access to all permissions assigned to junior roles
  - ✗ Difficult to implement separation of duty requirements on roles that have a common senior role
  - ✗ Semantics of inheritance make implementation of multi-level secure systems difficult using RBAC models

# Introduction – The Idea

- An alternative approach to permission inheritance
  - Each permission has an orientation
  - Define explicit and implicit assignment
- Consider consequences for the implementation of multi-level secure systems

# Introduction – Overview of Talk

- Define new model for permission inheritance
- Consider the correspondence between RBAC and MAC
- Define constraints that are required for new model to implement MAC
- Concluding remarks
  - Contribution
  - Future work

# The Basic Model

- Partially ordered set of roles  $R$
- User-role assignment relation  $UA \subseteq U \times R$ 
  - We say  $u$  is explicitly assigned to  $r$  if  $(u, r) \in UA$
  - We say  $u$  is implicitly assigned to  $r$  if there exists  $s$  such that  $(u, s) \in UA$  and  $r \leq s$
- Permission-role assignment relation  $PA \subseteq P \times R$ 
  - We say  $p$  is explicitly assigned to  $r$  if  $(p, r) \in PA$
  - We denote the set of roles explicitly assigned to  $p$  by  $R(p)$

# Permission Inheritance

- The set of permissions  $P$  is the disjoint union of  $P^+$ ,  $P^-$  and  $P^0$ 
  - $P^+$  is the set of up permissions
  - $P^-$  is the set of down permissions
  - $P^0$  is the set of neutral permissions
- If every permission is an up permission ( $P^- = P^0 = \emptyset$ ) then we have the usual permission inheritance semantics of RBAC

# Implicit Assignment

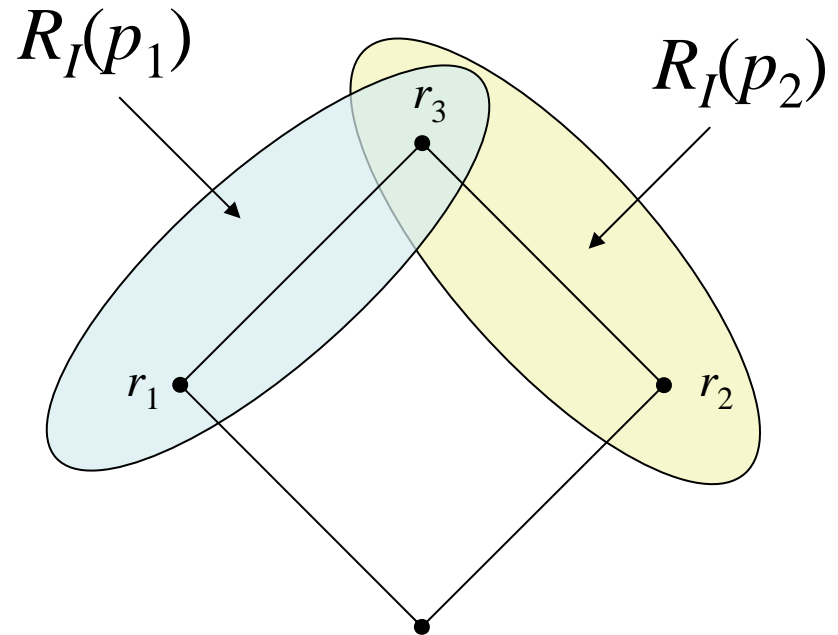
- Each permission  $p$  is implicitly assigned to a subset of  $R$  denoted  $R_I(p)$ 
  - If  $p \in P^+$  then  $R_I(p) = \{s \in R : \exists r \in R(p), r \leq s\}$
  - If  $p \in P^-$  then  $R_I(p) = \{s \in R : \exists r \in R(p), r \geq s\}$
  - If  $p \in P^0$  then  $R_I(p) = R(p)$
- We refer to  $R_I(p)$  as the **effective** roles of  $p$

# Permission Usage

- In RBAC a user  $u$  activates a session  $S \subseteq R$  by selecting a subset of the roles to which  $u$  is implicitly assigned
- A user  $u$  will be granted the use of permission  $p$  provided  $u$  has activated one of the effective roles of  $p$ 
  - In other words  $S \cap R_I(p) \neq \emptyset$

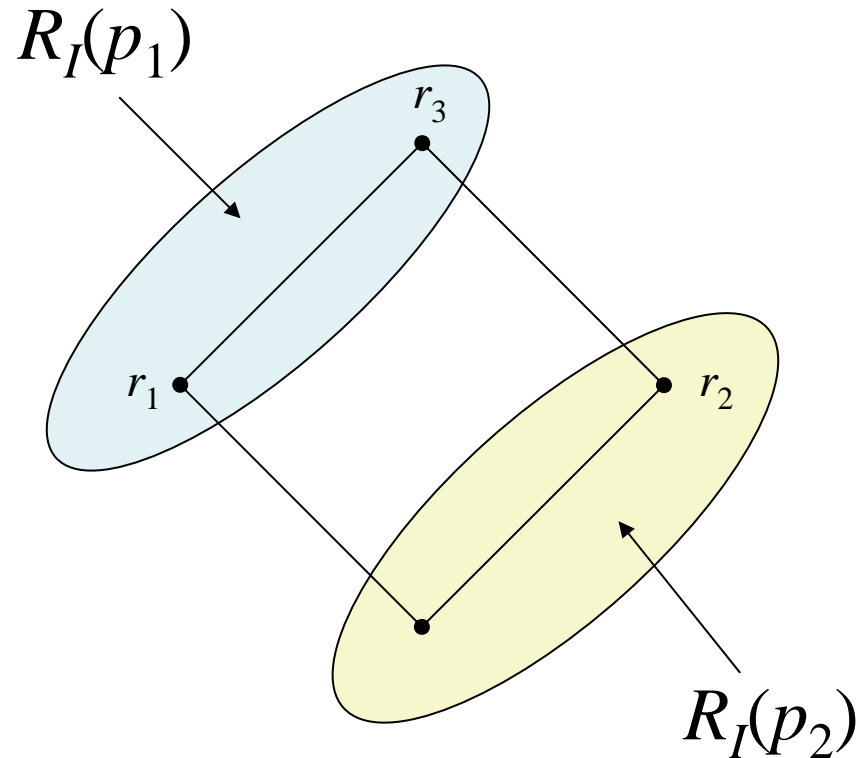
# Permission Usage

- $p_1, p_2 \in P^+$
- $(p_1, r_1) \in PA$
- $(p_2, r_2) \in PA$
- $u$  can make use of both  $p_1$  and  $p_2$  by activating  $r_3$ 
  - Standard RBAC interpretation



# Permission Usage

- $p_1 \in P^+$
- $(p_1, r_1) \in PA$
- $p_2 \in P^-$
- $(p_2, r_2) \in PA$
- $u$  must activate both  $r_1$  and  $r_2$  to make use of  $p_1$  and  $p_2$ 
  - Separation of duty constraints can be defined to prevent this if required



# Permissions

- A permission  $p$  has the form  $(o, M)$  where  $o$  is an object and  $M$  is a set of (access) methods
  - Typically the orientation of a permission will depend on  $M$
- We define  $p \leq q$  if  $p = (o, M)$  and  $q = (o, N)$  and  $M \subseteq N$ 
  - We write  $p < q$  if  $p \leq q$  and  $p \neq q$
- This is not the only model for permissions
  - A permission could have the form  $(O, m)$
  - A permission could have the form  $\{(o_1, m_1), \dots, (o_n, m_n)\}$

# Constraints

- If  $p < q$  then either  $p$  and  $q$  have the same orientation or  $q \in P^0$ 
  - The direction of inheritance is consistent with weaker permissions
  - This is essentially a consistency constraint
- If  $p < q$  then  $R_I(p) \not\subseteq R_I(q)$ 
  - A permission cannot be implicitly assigned to more roles than a weaker permission
  - This is essentially a redundancy check (what is the point of the weaker permission otherwise?)
- These constraints are applicable to any model where the set of permissions is partially ordered

# RBAC and MAC – Similarities

- Permissions
  - Semantics of read permission inheritance in standard RBAC are consistent with interpretation in MAC
- Security labels
  - Set of roles assigned to  $u$  can be regarded as the security label of  $u$
  - Set of roles activated by  $u$  in a session can be regarded as the current security label of  $u$
- The \*-property
  - Dynamic permission-based separation of duty

# RBAC and MAC – Differences

- Permissions
  - Write permissions are not consistent
    - Existing approaches use two different hierarchies
  - Usage is incompatible
    - In RBAC usage is based on existential criterion
    - In MAC usage is based on universal criterion
- Security labels
  - What is the security label of an object or of a permission?

# Simulating BLP – Permissions

- We assume there exists a set of access modes  $\mathcal{M} = \mathcal{M}_r \cup \mathcal{M}_w$ 
  - It is not necessarily the case that  $\mathcal{M}_r \cap \mathcal{M}_w = \emptyset$
- A permission of the form
  - $(o, \{m\})$  is called an **atomic** permission
  - $(o, M_r)$  is called a **read** permission
  - $(o, M_w)$  is called a **write** permission

# Simulating BLP – Permissions

- A permission is **simple** if it is a read or write permission and **compound** otherwise
- A permission is **minimal** (respectively **maximal**) if it is assigned to a role and any weaker (stronger) permission is not assigned to any role
  - If  $(o, \{m\})$  is assigned to a role then it is necessarily minimal
  - If a permission is minimal then it is assigned to a unique role

# Simulating BLP – Constraints

- $PA$  is a function (not a relation)
  - Every permission  $p$  is assigned to a unique role  $r$
  - The security level of  $p$  is  $r$
- For each object  $o$  there is a unique minimal read permission  $p_r$  and a unique maximal write permission  $p_w$ 
  - Hence there exists roles  $r_{\min}$  and  $r_{\max}$  such that  $(p_r, r_{\min}) \in PA$  and  $(p_w, r_{\max}) \in PA$
  - We define the security level of an object to be the range  $[r_{\min}, r_{\max}] = \{r \in R : r_{\min} \leq r \leq r_{\max}\}$

# Simulating BLP – Constraints

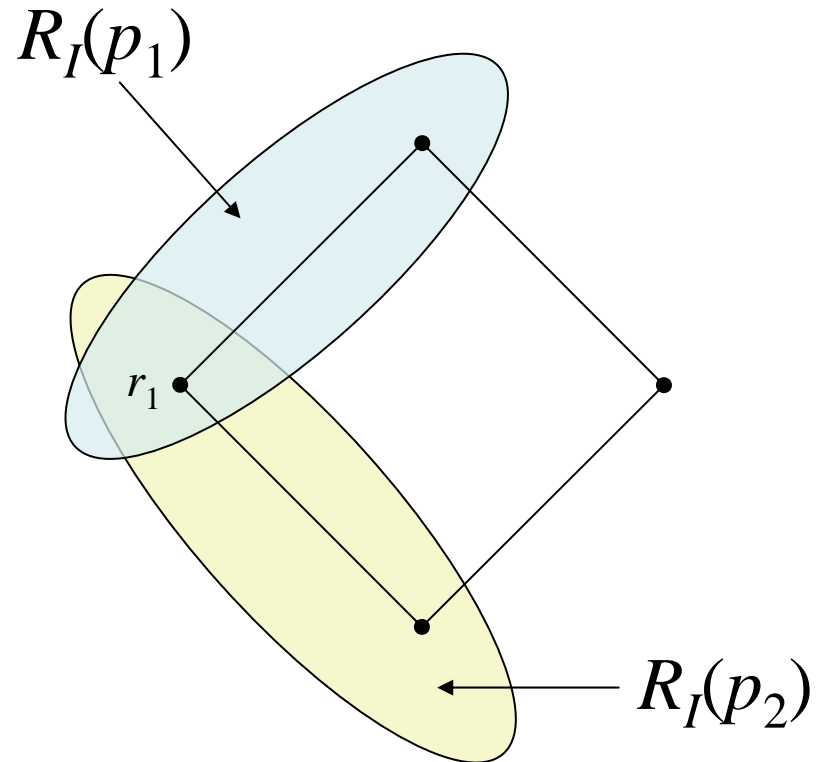
- If  $p < q$  then  $R_I(q) \subset R_I(p)$ 
  - The stronger the permission, the fewer the roles to which it is implicitly assigned
  - This is a stronger condition than the ones defined earlier
- If  $p$  is a write permission then  $p \in P^-$
- If  $p$  is a read permission then  $p \in P^+$

# Simulating BLP – Constraints

- If  $p$  is a compound permission then  $p \in P^0$ 
  - *A compound permission can only be assigned to a role within the security level of  $o$*
  - If  $r_{\max} < r_{\min}$  for  $o$  then a compound permission cannot be assigned to any role
    - An audit log file which low level users (must be able to) write to and which only certain high level users can read but no user should be able to both read and write

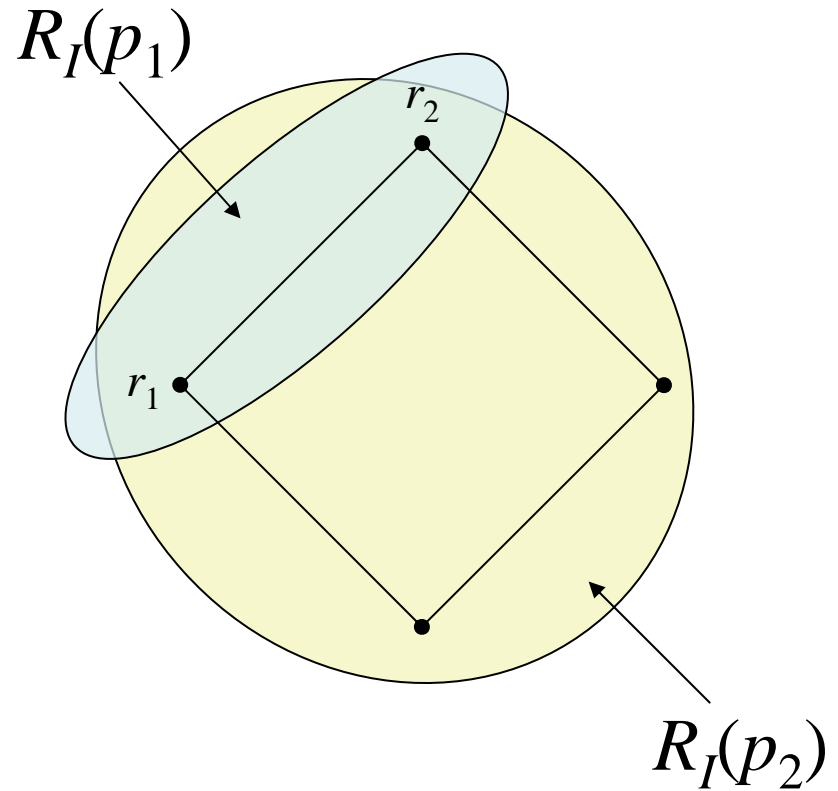
# Simulating BLP – Example

- $\mathcal{M} = \{r, w\}$
- $p_1 = (o, \{r\})$
- $(p_1, r_1) \in PA$
- $p_2 = (o, \{w\})$
- $(p_2, r_1) \in PA$
- Security level of  $o$  is  $\{r_1\}$ 
  - $p_3 = (o, \{r, w\})$
  - $p_3$  can only be assigned to role  $r_1$
  - Corresponds to security level in BLP



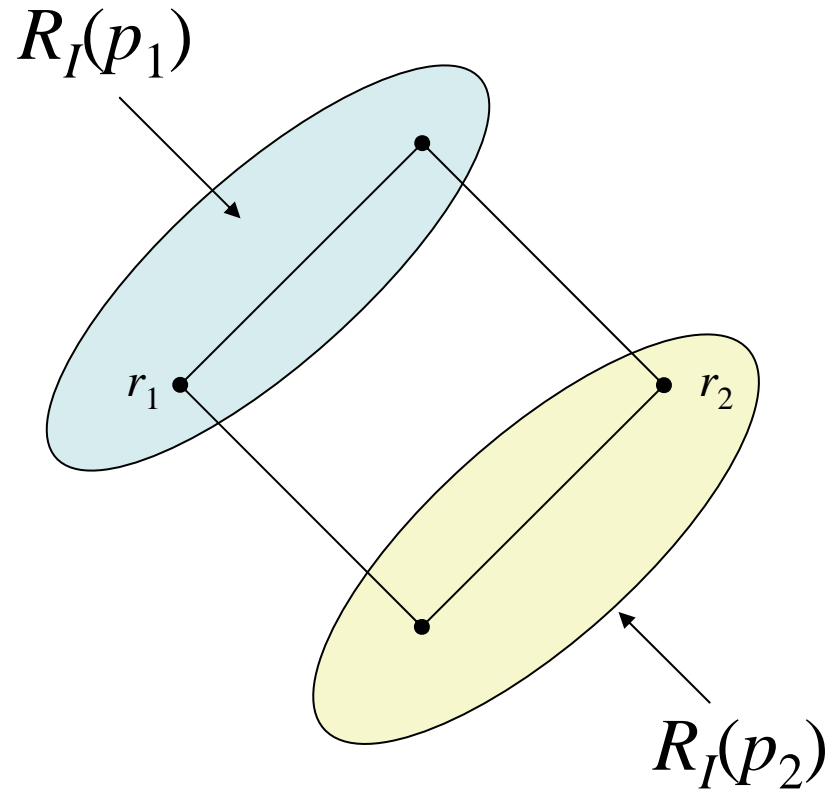
# Simulating BLP – Example

- $\mathcal{M} = \{r, w\}$
- $p_1 = (o, \{r\})$
- $(p_1, r_1) \in PA$
- $p_2 = (o, \{w\})$
- $(p_2, r_2) \in PA$
- Security level of  $o$  is  $\{r_1, r_2\}$ 
  - $p_3 = (o, \{r, w\})$
  - $p_3$  can be assigned to  $r_1$  or  $r_2$



# Simulating BLP – Example

- $\mathcal{M} = \{r, w\}$
- $p_1 = (o, \{r\})$
- $(p_1, r_1) \in PA$
- $p_2 = (o, \{w\})$
- $(p_2, r_2) \in PA$
- Security level of  $o$  is  $\emptyset$ 
  - $p_3 = (o, \{r, w\})$
  - $p_3$  cannot be assigned to any role



# Contributions

- Permission-based separation of duty is now possible
  - $R_I(p) \cap R_I(q) = \emptyset$
- Our approach offers more flexibility than standard RBAC models
  - Can assign a permission to a role and propagate that to less senior roles
    - May be useful for assigning permissions to a tree-like directory structure
  - Our approach does not introduce any significant overheads to permission usage checking
- Provides a simple way to implement multi-level properties using RBAC
  - Previous approaches have typically required the use of a second hierarchy and cannot cope with compound permissions

# Future Work

- Simple Security Theorem
  - Define state transitions
  - Define administrative framework (covered in proceedings)
- We still have the problem that any user assigned to a top role can access all permissions ...
  - ... although the problem is less acute because the effective set of roles for a permission means that the user may have to activate a less senior role to make use of it
    - The principle of least privilege is more accurately implemented
  - Assign a “floor” to each user
    - A floor partitions the role hierarchy
    - A user can only activate roles between the floor and the roles to which he is explicitly assigned