[DRAFT] Role-Based-Access-Control (RBAC) Permission Notes

Perforce Professional Services

Version v2025.1, 2025-10-29

Table of Contents

Todo Items

- Add scripts to demonstrate reporting RBAC permissions (likely using P4PHP and a webserver). This allows some experimentation with RBAC without changing P4d.
- Add triggers or new broker functions to handle RBAC permissions. These could simply validate RBAC rules are followed.

DRAFT NOTICE

This document is in DRAFT status and should not be relied on yet. It is a preview of a document to be completed in a future release.

Preface

This document discusses some ideas on managing large complex Perforce permissions schemas using Role-Based-Access-Control (RBAC) methods.

Perforce has a simple model of permissions utilizing users, groups, nested groups, and permissions. While this structure is very flexible, it sometimes may get disorgnized when working with large complex permission schema's. This document proposes some additinal formalization of the existing p4 groups into a RBAC model consisting of `user groups` and `role groups`, along with rules for their useage.

Motavation: Many large Perforce sites often look for improved methods in managing Perforce permissions. Some sites manage all user/groups/permissions in an external tool and generate / import the entire protections table into P4d. Some sites manage all groups externally in `MS Active Directory` groups synced to Perforce. However, these methods often lead to

This document adds some additional formal structure to the existing P4d permission model simliar to *RBAC* methods used with *Microsoft Active Directory*.

The goal in these examples is to configure three LVM storage volumes separate from the OS volume on a server destined to become a Helix Core server. At the start of this procedure, empty volumes with no data are formatted.

Please Give Us Feedback

Perforce welcomes feedback from our users. Please send any suggestions for improving this document or the SDP to consulting-helix-core@perforce.com.

In particular user groups are often found with nearly identical permissions assigned but no clear reason why

Chapter 1. RBAC Intro

Role-based-access-control (RBAC)

> RBAC, or Role-Based Access Control, is a security method that restricts system access based on a user's role within an organization, rather than on their individual identity. It works by assigning permissions and privileges to roles, which are then assigned to users, ensuring that individuals only have access to the data and applications needed for their specific job function

1.1. What is RBAC ?

RBAC is a formalized method of controlling permissions assigned to users in a more structured way.

Instead of assigning permissions to users or groups in an ad-hock mannner, we we assign Users to User Groups, User Groups to Role Groups, and Role Groups to Permissions.

We specifically avoid assigning Permissions directly to user. We also avoid any nesting of User Groups within other User Groups.

```
Assignment Hiearchy:
Users->UserGroups->RoleGroups->Permissions
```

1.1.1. Sample Schema

For this example we use a Game Developer company that uses internal staff and external contractors to fill many of the same roles. The internal and external staff are kept separated, but may share many of the same roles. The GameEngine code contains modification shared across several projects, so the folks editing the GameEngine are a subset of Developers.

This is a sample schema for **Project1** containing several *Roles* and several *User Groups* .

Project1: Roles

Note that each Role gets permissions assigned.

- Proj1_CodeDev_Role - Development of the Game
- Proj1_GameEngineDev_Role - Modification of the Game Engine
- Proj1_Artist_Role - Art, 3d Models
- Proj1_AudioEngineer_Role - Audio tracks
- Proj1_BuildEng_Role - Build automation

Project1: User groups

Note that no permissions may be assigned, only roles. External groups are explicitly separated from Internal. Typical assigned roles are shown.

- Team1_Developers
  - Proj1_CodeDev_Role
  - Proj1_GameEngineDev_Role
- Partner1_Dev_External
  - Proj1_CodeDev_Role
- Team1_Artists
  - Proj1_Artist_Role
- APAC_Art_External
  - Proj1_Artist_Role

1.1.2. Sample Report

Ultimately for this project, we could procedue an RBAC formatted representation of each project's permissions.

1.2. Problems RBAC Solves

Some of the issues commonly found with existing ad-hock permissions assignements are.

1. Discrete permissions are assiged directly to a user, leading to a collection of permissions without any documentation of what each permission or group of permissions are for. Only the Subject Matter Experts (SME's) understand the permissions, not the IT folks assigning new users to a group.
2. User groups are sometimes nested to pick up permissions of the parent group, however this relationship is not often well displayed or understood and often leads to over assigning permissions. In general we avoid the pernicious nesting of user groups within user groups.
3. Requests to add permissions are often (give new user X the same permissions as Y) without any clear understanding of what those permissions are.

Microsoft guidance on Nesting of groups in Active Directory: https://www.techtarget.com/searchwindowsserver/tip/Active-Directory-nesting-groups-strategy-and-implementation This "ADGLP" strategy allows 2 levles of nesting, but since we don't have a global context and are dealing with a single project inside a Perforce instance our RBAC limits nesting to just *User Groups* as subgroups of *Role Groups* and no further nesting.

1.3. Scanerios where RBAC may be beneficial

Adopting Perforce **RBAC** procedures as described here may be beneficial if.

1. Folks requesting permissions for new users don't typically know the exact list of permissions required. Often requests are made to *provide permissions same as user X*. **RBAC** gives names (e.g Roles) to groups of permissions making it easier for folks to understand

2. Perforce permissions are often assigned by IT folks who are not Perforce experts. This follows closely how Active Directory Roles are assigned for multiple corporate Roles, without IT knowing the detailed permissiongs granted by the Roles. Perforce experts are involved with creating or changing permissions in the Roles, but assigning them.
3. There are user groups in Perforce with very similar assignments of permissions , but no clear understanding of why the differences exist.

Todo Items

1. Show a sample RBAC report output. See [Sampe Schema].
2. Need a demo script that reports a basic RBAC Schema report. This could be simple command-line canned report, or a php App.

Chapter 2. Implementation

2.1. Current status

Currently there is no built-in suppportin P4d for *RBAC* roles, however these procedure use existing p4d functionality by simply reporting *RBAC* permissions. Permission changes in an *RBAC* enabled project that don't meed the rules, are flagged or ignored. Some of these ideas could be supported futher with triggers or broker scripts, especially if further interest is expressed.

Design note

The implementation discussed here uses procedures or scripts only to report the Perforce Permissions in an enhanced RBAC Schema. The *RBAC* schema has slightly stricter rules, but uses existing p4d commands. The design goal is that these procedures do not intercept normal P4d permission operations. That is, initally these procedures are designed to be fail-safe by simply reporting permission in an *RBAC* manner, and simply reporting any *RBAC* schema violations. Later, enforcement of *RBAC* schema could be added, but could easily disabled if no longer needed.

2.2. Supporting Role Groups

To support the concept of Role groups we split the namespace of User groups into the following:

User Groups

- *User groups* contain only users and can't be nested inside other user group.
- *User groups* are never assigned permissions directly
- *User groups* may be assigned as a subgroup of a *Role Group* to pick up permissions

Role Groups

- *Role Groups* never contain users
- *Role Groups* are assigned permissions
- *Role groups* have subgroups containing *User Groups*

So in effect we are enforcing some new procedures.

1. Permissions are never directly assigned to a user or group. We enforce **ALWAYS** creating a **Role** for any single permission or group of permissions. This Role should have an clear name that describes it's function. This is an *extra* layey of inderection being imposed.
2. Other than a single level nesting of *User Groups* as subgroups of *Role Groups* there is no other nesting allowed. *User Groups* may not be nested inside other *User Groups* as that is pernicious.

By following these rules, the permission schema is simplified so that most folks only ever need to understand the limited set of *Roles* for the project. Only the Project SME's need to understand the permissions assigned to a *Role* and rarely change them.

2.3. Reporting

The simplest reporting can be accomplished by command-line p4 scripts to report some of the following:

- List Role groups
  - List Role group and associated User (sub-groups), and effective permissions assigned.
- List User Groups
  - List User Groups and associated Roles (and optionally a list of users assigned)
- List users in the project (belonging to at least one User Group in the project) and their assigned Roles

Appendix A: DRAFT NOTICE

This document is in DRAFT status and should not be relied on yet. It is a preview of a document to be completed in a future release.