Table of Contents

1 Document History..................................................................................................................................1
2 Introduction............................................................................................................................................2
3 Assumptions........................................................................................................................................2
4 Scope......................................................................................................................................................2
5 Definitions..............................................................................................................................................2
6 Requirements........................................................................................................................................5
   6.1 Security...............................................................................................................................................5
       6.1.1 Consolidate risk: Essential.................................................................5
       6.1.2 Encapsulate risk: Essential.................................................................5
       6.1.3 Auditable: Essential.............................................................................5
       6.1.4 Explanatory: Aspirational.................................................................5
   6.2 Efficiency........................................................................................................................................6
       6.2.1 Minimize database access: Essential...............................................6
       6.2.2 Minimize necessary API calls: Essential...........................................6
       6.2.3 Inheritance: Essential.........................................................................7
       6.2.4 Caching: Needs investigation.............................................................8
   6.3 Comprehensiveness, self-containment, generality.................................................................8
       6.3.1 Store all access control data: Essential...........................................8
       6.3.2 Perform all access control processing: Essential.............................8
       6.3.3 Access type implication processing: Essential...................................8
       6.3.4 Access type implication storage: Desirable........................................9
       6.3.5 Multi-granular: Essential.................................................................9
       6.3.6 “All” and “site” entities: Essential....................................................9
       6.3.7 Generic/Agnostic: Essential.............................................................9
       6.3.8 Roles: Essential..................................................................................10
       6.3.9 Prioritization: Essential.................................................................10
       6.3.10 Configurable/Tunable: Desirable....................................................10
   6.4 Simplicity.......................................................................................................................................11
       6.4.1 Encapsulate access control debugging and testing: Essential........11
       6.4.2 Simple for most API callers: Essential.............................................11
       6.4.3 APIs for simple User Interfaces: Essential.......................................11
7 API Caller Examples............................................................................................................................12

1 Document History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Comments</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 24, 2006</td>
<td>1</td>
<td>Initial version, retrospective and taking into</td>
<td>Matt Oquist</td>
</tr>
</tbody>
</table>

1
2 Introduction

The requirements for an access control system are driven by the requirements for a digital portfolio system; see the companion document “SPDC Digital Portfolios Project Portfolio Requirements” for further information about the portfolio system requirements. The portfolio system requires that it is “possible” and “intuitive” to publish portfolio artifacts, and it requires that some users can be “advisers” who are able to review and comment on portfolio artifacts. An access control system capable of meeting at least these requirements must be implemented.

3 Assumptions

- The access control system will be licensed by the GNU General Public License version 2.
- The access control system will be implemented within Moodle.
- Moodle should only have one access control system (or any of the synonymic systems: “permissions”, “roles & capabilities”, etc.).

4 Scope

The scope of the access control system is to provide a single, centralized, extensible system for access control to any and all objects at all levels of granularity within Moodle. The design of the first implementation should take into account all feasible future access control needs, so that subsequent developments in Moodle will necessitate as little redundancy, confusion, and development inefficiency as possible.

5 Definitions

1. Access control

   This refers to a mechanism by which some access to an entity may be controlled. Access control is taken to include the concept of ownership. The following terms are taken to denote the same general set of concepts:
   
   1. Access control
   2. Roles & Capabilities
   3. Permissions [management]

2. Access control relation

   This is a more general case of an access control specification, which refers either to
ownship of a target entity by a designated entity, or to the specification that a designated entity is granted or denied access to a target entity.

3. Access control specification

This is a specific type of access control relation which specifies a particular type of access to a target entity that is granted or denied to a designated entity. The following are examples of access control specifications:

<table>
<thead>
<tr>
<th>Designated Entity</th>
<th>Access Type</th>
<th>Granted/Denied</th>
<th>Target Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>user “euthyphro”</td>
<td>read</td>
<td>granted</td>
<td>course “Wisdom101”</td>
</tr>
<tr>
<td>user “socrates”</td>
<td>course creation</td>
<td>granted</td>
<td>site</td>
</tr>
<tr>
<td>user “einstein”</td>
<td>convert</td>
<td>granted</td>
<td>matter</td>
</tr>
<tr>
<td>user “einstein”</td>
<td>convert</td>
<td>denied</td>
<td>energy</td>
</tr>
<tr>
<td>teachers of “History 101”</td>
<td>edit</td>
<td>granted</td>
<td>repository “History archives”</td>
</tr>
<tr>
<td>group “Vermicious Knids”</td>
<td>write</td>
<td>denied</td>
<td>(all)</td>
</tr>
<tr>
<td>&lt;subsystem&gt; instance 12</td>
<td>moderate</td>
<td>granted</td>
<td>forum instance 17</td>
</tr>
<tr>
<td>enrollees in “Science 101”</td>
<td>write</td>
<td>granted</td>
<td>science project wiki</td>
</tr>
<tr>
<td>user “lamarck”</td>
<td>write</td>
<td>denied</td>
<td>science project wiki</td>
</tr>
<tr>
<td>user “donquixote”</td>
<td>adviser</td>
<td>granted</td>
<td>user “sanchopanza”</td>
</tr>
<tr>
<td>user “sanchopanza”</td>
<td>evaluate</td>
<td>granted</td>
<td>user “asop”</td>
</tr>
<tr>
<td>user “asop”</td>
<td>adviser</td>
<td>granted</td>
<td>all students</td>
</tr>
</tbody>
</table>

4. Access type

This refers to a type of access that may be in an access control specification. Examples include: read, write, create, remove, moderate, edit, move, copy.

5. Aspirational

Aspirational requirements would be nice to have.

Aspirational requirements may be met.

6. Deny, denied, denying, etc.

This refers to an access control specification whereby the specified access is denied (or “disallowed”) to the designated entity.

7. Designated entity

This refers to any entity which can be granted or denied access to another entity. A user is a typical case, though any entity, such as a file, might be a designated entity.

8. Desirable

Desirable requirements “round out” the access control system into one that is not only
sufficient, but attractive and “fully-featured”.
Desirable requirements should be met.

9. Essential
The meeting of essential requirements defines the access control system; any design of an access control system for Moodle that does not meet these requirements is insufficient.
Essential requirements must be met.

10. Grant, granted, granting, etc.
This refers to an access control specification whereby the specified access is given (or “allowed”) to the designated entity.

11. May
See “aspirational”.

12. Meta-access-control and meta-permission
This refers to an access control specification that concerns the designated entity’s permission to create or remove other access control specifications. For example, a department head may have permission to grant other teachers permission to create courses.

13. Must
See “essential”.

14. Needs investigation
Requirements with this designation may or may not be feasible and will require further investigation during the design phase.

15. Ownership
This is a type of access control relation in which one entity is said to “belong to” the other. For example, a user is generally taken to “own” the files in her home directory in this sense.

16. Role, roles
This refers to the name of a function or position that one or more designatable entities may fill. For example, a user could be an “adviser”, a “parent”, a “manager”, a “weekly seminar leader”, a “mentor”, a “visitor”, a “former student”, a “librarian”, a “teacher”, a “secretary”, a “teaching assistant”, etc.

17. Should
See “desirable”.

18. Target entity
This refers to any entity to which access can be granted or denied. A file is a typical case, though any entity, such as a user, might be a target entity.
19. UI

This is an acronym for “user interface”.

20. Unspecified [access]

This refers to any case in which a given designated entity has no access control specification for a given target entity. That is, the access in question has been neither granted nor denied.

6 Requirements

6.1 Security

6.1.1 Consolidate risk: Essential

All the security risks related to access control must be consolidated as much as possible into one central body of code and one central set of database tables. This code and these tables can then undergo scrutiny for security issues.

6.1.2 Encapsulate risk: Essential

All the security risks related to access control must be encapsulated as much as possible from the rest of Moodle in order to limit the risk of security problems from improper, unclear, or abused coupling between the access control system and its external API callers.

For example:

- No external API caller may read or modify access control database tables directly.
- No external API caller may respond inappropriately to the results of an access request or an access check. (If the access control system says access is denied, the external API caller must not allow access to proceed.)

6.1.3 Auditable: Essential

The access control system must provide a mechanism by which access requests and changes are logged and available for subsequent review.

6.1.4 Explanatory: Aspirational

The access control system may provide a mechanism by which processing of access queries for a given target entity and designated entity can be “explained” in human-readable form. Any access control system can yield results that surprise the system administrator; a user may have access to an entity when the administrator has tried to deny that access, or vice-versa. An explanatory mechanism to show the administrator why or how a designated entity has a specified access to a target entity could be useful.
6.2 Efficiency

6.2.1 Minimize database access: Essential

The access control system must minimize the number of queries and updates to the access control database tables. Whenever possible, a single API call should result in no more than a single database read and/or a single database update.

6.2.2 Minimize necessary API calls: Essential

The access control system must allow a single API call to address multiple target entities, multiple designated entities, multiple access types, etc. This is essential because several core elements of Moodle must be able to determine quickly and efficiently what kind(s) of access the current user has to hundreds, if not thousands of target entities.

The following scenario illustrates why it is desirable to minimize necessary API calls:

• User “danielle” is enrolled in the course LIT101.
• User “glenda” is a teacher for the course LIT101.
• User “saeko” is a student-teacher for the course LIT101.
• User “sidsel” is danielle's adviser.
• Course LIT101 has hundreds of resources and activities, including text/web pages, assignments, chats, forum discussions, glossaries, lessons, quizzes, surveys, wikis, etc.
• Activities and resources may be “hidden”, which means that regular course enrollees do not have access.
• A student (or group of students) may be granted some type of additional access to a given activity or resource. For example, a student (or student group) may be responsible for editing a weekly quiz activity.
• When any user loads the course page, the course code must call access control APIs to determine which of these resources and activities the current user has access to, and what kinds of access she has.

➔ Regardless of which user loads the course page, it would be very inefficient to iterate through all the course resources and activities, checking for the current user's access to each one. Instead, it must be possible to construct a single data set that can be passed to a single access control API, which will return a single data set containing all the necessary results of the queries.

1 This is really an design/implementation note, but it is likely that the developer(s) of the access control system will have helpful debugging mechanism to give them this type of information, so the debugging mechanism could perhaps be cleaned up into a human-readable form for use by the site administrator.
6.2.3 **Inheritance: Essential**

The access control system must be capable of storing, processing, and interpreting arbitrary inheritance relations among target and designated entities.

The following are examples of such inheritance relations:

1. a portfolio artifact may inherit access control specifications from its parent portfolio(s)
2. a course activity may inherit access control specifications from the course

Given such specified inheritance relations, the access control system must be able to evaluate whether a designated entity has access to a target entity when access is granted only to a "parent" (or "grandparent", etc.) of the targeted entity.

The following scenario shows why this relational storage and processing in the access control system is essential for the sake of efficiency:

- User “danielle” is enrolled in the course LIT101.
- User “glenda” is not yet enrolled in the course LIT101.
- Course LIT101 has hundreds of resources and activities, including text/web pages, assignments, chats, forum discussions, glossaries, lessons, quizzes, surveys, wikis, etc.

➔ When glenda is enrolled in LIT101, there must be one new access control specification [for the entire course], and not hundreds of new access control specifications to account for every resource and activity in the course.

➔ If an external API caller queries the access control system to discover whether danielle has [normal, enrolled student] access to a particular activity (“Wizards Quiz”) in LIT101, the access control system must be able to determine that though there may be no access control specification with danielle as the designated entity and Wizards Quiz as the target entity, there is an access control specification that danielle has “enrolled student” 2 access to the course, the Wizards Quiz inherits this specification, and danielle has been granted “enrolled student” access to the Wizards Quiz.

Alternatively, it would be possible for the external API callers to do [some of] this themselves. For example, the course code could discover that the current user is enrolled in “Farming 101” and “decide” or “know”, without consulting the access control system, that the current user therefore has access to all the course activities and resources. Such an implementation is limiting because the current user may access one of the Farming 101 activities outside of the course context (for example, the user may view a general list of her submitted assignments), and the other context will then have to duplicate the logic of the course implementation in order to ensure that the same access is provided. Additionally, such an implementation obviates the value of

---

2 “Enrolled student” access is the example rather than “read” or “write”, because an enrolled student may have "read" access to some course entities, and "write" to others, etc. The course code will have to determine what names it will give the types of access it will use.
the following requirements:

- (6.1.1), above
- (6.1.2), above
- (6.3.1), below
- (6.3.2), below
- (6.4.1), below

6.2.4 Caching: Needs investigation

The access control could potentially cache the results of past access queries and
requests in system memory and thereby avoid unnecessary repetition of database
access and data processing. Cache coherency would be a major stumbling block, and
investigation during the design phase will be required to determine if this is feasible
and would be profitable.

6.3 Comprehensiveness, self-containment, generality

6.3.1 Store all access control data: Essential

The access control system must be exclusively responsible for the storage of all data
related to access controls. This includes meta-access-control data identifying
designated entities that are able to create and remove other access control
specifications.

This is essential for the access control system to be self-contained.

6.3.2 Perform all access control processing: Essential

The access control system must be exclusively responsible for retrieving access
control data from the database, processing access control data, interpreting access
control data, and updating access control data in the database.

This is essential for the access control system to be self-contained.

6.3.3 Access type implication processing: Essential

Section (6.2.3), above, is related.

The access control system must be capable of processing specified relations among
access types in which one access type implies another.

For example, an external API caller may wish to specify that a grant of 'write' access
implies a grant of 'read' access.

Note that such an implication must hold only when the caller specifies, however.
Different callers may specify different implications. See section (6.3.7), below.
For example, external API callers in the repository code may specify that 'write' access implies 'delete' access, but the assignment activity module may not want a user's 'write' access to her own assignment submission to imply 'delete' access to the same.

### 6.3.4 Access type implication storage: Desirable

The access control system should be capable of storing the type of specified relations discussed in section (6.3.3), above. Stored implication relations may be referenced only when the caller specifies, because different callers may specify different implicatory relations and the access control system can make no assumptions about such relations. See section (6.3.7), below.

### 6.3.5 Multi-granular: Essential

The access control system must be able to handle target entities at all levels of granularity. Examples of target entities include:

- course, metacourse
- assignment, assignment submission
- forum, forum discussion, forum post
- resource
- activity
- module
- the entire site
- blog, blog post, blog post comment
- sitewide portfolio configuration, portfolio, portfolio artifact, portfolio artifact comment

### 6.3.6 “All” and “site” entities: Essential

A user may wish to give read access to her portfolio to “all” users (including “guest” or even no login), or perhaps to “all site users” (authenticated users). An administrator may wish to lock a user out temporarily by explicitly denying “all” types of access to “all” target entities. These and other scenarios such as these illustrate why the access control system must have a way to represent “all” entities, “all” access types, and “all site users” as an entity.

### 6.3.7 Generic/Agnostic: Essential

The access control system must be generic, which implies that it must be entirely agnostic about the nature of the access control specifications it stores and processes. This, in turn, implies that

- all access types are specified by the external API callers, and the access control
system is only aware of relations between access types when the external API caller specifies them

- all target entity types and designated entity types are specified by the external API callers, and the access control system is only aware of relations between these types when the external API caller specifies them

6.3.8 Roles: Essential

The access control system must be capable of storing and processing data relating designated entities and roles. This mechanism may then be used, for example, to place a particular user in a “school administrator” role, which will give the user the correct set of access control specifications for any “school administrator”.

6.3.9 Prioritization: Essential

The access control system must have a prioritization mechanism by which it can resolve conflicting access control specifications. For example, user “danielle” may be enrolled in the course “Making Meanings: Hegel and Berger”, which has a forum entitled “Epistemology: Outside-in or Inside-out?” Suppose danielle has misbehaved in the forums and the course instructor wishes to ban danielle from posting; the access control system must have a mechanism to store and process this ban at a higher priority than the specification that danielle has access to post in the forum by way of course enrollment.

Alternatively, suppose there is an entire group of students who have misbehaved, and the entire group has been banned from participation in scores of carefully-picked activities in a course; but the decision is made later that one member of the group should again be granted access to one of the activities. The access control system must have a mechanism to store and process this access grant at a higher priority than the pre-existing denial.

6.3.10 Configurable/Tunable: Desirable

The access control system should expose any configurable settings to the administrator, while assuming reasonable defaults that should suffice for most circumstances. Some of these configurable settings may aid the administrator in fine-tuning the access control system for greater efficiency given a particularly dominant application of the access control system. For example, a particular site may have users who routinely have thousands of small files stored in a repository, and the access control system may be tunable to treat and cache the most common requests more efficiently.

---

3 “When” implies here that the external API caller must specify such relations in every API call. Section (6.3.3), above, discusses the possibility of storing such relations, though even if these relations are stored, in every API call the caller would still need to identify that the stored [set of] relation[s] is to be referenced by the access control system.
6.4 Simplicity

6.4.1 Encapsulate access control debugging and testing: Essential

The access control system must encapsulate all debugging and testing of access control data storage and processing away from the rest of Moodle. Centralizing the access control code base provides a focal point for debugging and testing.

6.4.2 Simple for most API callers: Essential

The access control system must strive to provide a simple API that provides the major, central features of access control to the external API callers. Additional efficiency and features may be available via more complex APIs, but the simplicity of this subset of the total access control API will help to avoid confusion on the part of developers, thus promoting safe and secure usage of the access control system by external API callers.

6.4.3 APIs for simple User Interfaces: Essential

The access control system must provide an API that presents, and processes input from, a User Interface that can be used to manipulate arbitrary access control relations. External API callers may then use this API whenever the user will need to manipulate access control relations, including ownership and access control specifications. The default UI must be simple and intuitive, whereas more complexities and greater functionality may be presented to advanced users who select (or configure themselves to see) an “advanced” interface.

6.4.3.1 API to provide a UI to change ownership relations: Essential

The ownership changing user interface should be capable of handling ownership relations between any two types of entities. For example, a user may own a file, a group may own a portfolio, or a course may own a forum.

6.4.3.2 API to provide a UI to change which designated entities have a given access to a specified target entity: Essential

This is a target-entity-centric UI. For example, this type of UI will typically allow users to determine who is granted a specific type of access to a specific target entity. The following screenshot is an example of this, allowing the Moodle administrator to determine which users are granted admin access to the Moodle site:

---

4 Having more than one layer of “advanced” should not be out of the question. It is easy to imagine 1) a basic interface, 2) an “advanced” interface with a few extra knobs, and 3) a “very advanced” interface with lots of warnings, but which would allow the administrator to configure access controls in general, without clicking through courses, activities, resources, etc. to reach the relevant access control user interfaces. Levels 1 and 2 may only allow manipulation of designated entities such as users, groups, and courses, whereas level 3 would allow manipulation of entity pairs of any type. Note the related discussion in section (6.4.3.1), above. The level 3 interface would probably be an extended version of the one discussed in section (6.4.3.3), below.
6.4.3.3 API to provide a UI to change which target entities (of a specified class) a specified designated entity has a given access to: Essential

This is a designated-entity-centric UI. For example, the Moodle administrator could use it to modify which forums a given user has access to “moderate”, or which teachers a given student has access to “review”.

7 API Caller Examples

- Forum module needs to verify a user's permission to read or post within a given discussion
- Repository needs to verify a user's permission to view the list of files in a folder, to upload a new file, to delete a file, to edit a file, to publish a file, etc.
- Course needs to discover what access the current user has to each course element (resource, activity, etc.)
SPDC Digital Portfolios Project: Access Control Requirements

- Site needs to discover which courses the current user has access to view, access to enroll in, access to delete, access to rename, etc.
- Site needs to present a UI for the site administrator to assign other site administrators.
- Forum needs to present a UI to allow a moderator to assign other moderators.
- Repository needs to present a UI to allow an administrator to change the ownership of a file to a different user, group, course, forum, etc.
- Portfolio needs to present a UI to allow a user to publish her portfolio to the world, to a specific group, or a specific user.