Discretionary Access Controls for a Collaborative Virtual Environment

Timothy Wright[1] and Greg Madey[2]

[1]The Center for Research Computing, University of Notre Dame – ITC, Notre Dame, IN, 46556, United States
[2]Department of Computer Science & Engineering, University of Notre Dame, Notre Dame, IN, 46556, United States

Abstract—**As collaborative virtual environments (CVEs) are more widely used, participant access to CVE objects and information becomes a significant concern. In virtual reality games, storefronts, classrooms, and laboratories, for example, the need to control access to spaces and objects is integral to the security of activities taking place there. However, limited access controls are typically available in CVEs. Often, such controls are course-grained, only protecting against movements by unauthorized participants into specific areas. In answer to this deficiency, we offer a discretionary access control (DAC) system based on traditional concepts of users and groups, and tailored to the needs of a CVE. Our system, called WonderDAC, includes the ability to restrict movement into areas, as well as control interactions with objects. A basic WonderDAC prototype has been implemented within the Project Wonderland CVE.**

Index Terms—**Virtual reality, Collaborative virtual environment, Access control, Discretionary access**

I. INTRODUCTION

We take the term "collaborative virtual environment" (CVE) to mean a graphical, multi-user, virtual reality environment capable of operating on a typical, modern workstation. Often such environments are associated with game playing (e.g., the Quake series of games, massively multiplayer online role-playing games [MMORPGs] such as World of Warcraft), though commerce, educational, research, and social networking systems are also being realized through CVE technology [1-4].

The essential quality of any CVE is, of course, the collaboration among participants; however, this can also serve as an avenue for problems. Players in a game may elect to cheat, patrons of a virtual store might try to steal, researchers could inadvertently damage information or interfere with an experiment, and participants of a social networking system might have sensitive, personal information exposed. Problems of this ilk have manifested themselves in commercial CVEs such as Second Life. For example, the December 2006 malicious disruption of a CNET interview taking place in Second Life [5], the March 2007 defacement attack carried out on John Edwards' presidential campaign headquarters [6], the April 2007 attack on Toyota's Scion storefront [7], or the myriad of other, similar attacks and disruptions caused by miscreants (also called "griefers") in Second Life [8, 9]. Malware exploits within CVE worlds are also a concern, as demonstrated by the November 2007 QuickTime vulnerability within Second Life: an attacker could leverage a bug in the QuickTime player to "crash or exploit the Second Life viewer [10]." Through the presence of better privacy and integrity controls, incidents and weaknesses such as these could have been substantially mitigated, if not outright prevented.

To manage the integrity and privacy risks of a CVE, we note that security works best in layers, and that there are at least three layers where controls should be employed. The use of layered security controls, or *defense-in-depth*, is a well-understood means to reduce the chances of malicious and accidental damage to information systems [11-13]. The basic concept is that multiple controls can operate in a synergistic fashion, reducing risk more broadly as a group than as individual measures. Moreover, if one control becomes disabled or compromised, the other controls will still be in effect and may compensate.

For expedience in discussing access control, as well as for alignment with our search of the literature, we choose to divide a CVE into the following convenient layers: network communications, in-world access, and user experience. The network communications layer includes those mechanisms that transmit data between a CVE client and server, or among CVE peers (depending on whether a client-server or peer-to-peer architecture is employed). The in-world access layer encompasses course-grained access to a CVE system, its map, and resources. Finally, the user experience layer deals with the immersive qualities of a CVE, including the sense of awareness and privacy imparted by a virtual world's architecture, and a participant's virtual ability to see and hear. Fig. 1 depicts these layers.

We choose to focus on security in the middle layer for two reasons. First, network layer security is already an area of long-standing, significant research. Regardless of whether a CVE or some other system resides on top of this layer, the options and techniques to secure network communications are the same. Second, although there is no strong coupling between the network communications and in-world access layers, there is between the top two layers. In particular, the means of authentication and authorization used for the

Manuscript Received on 30 March 2009.
E-Mail: twright@nd.edu

in-world access layer should be leveraged by the user experience layer, too. Hence, this dependency compels us to deal with the in-world access layer first.

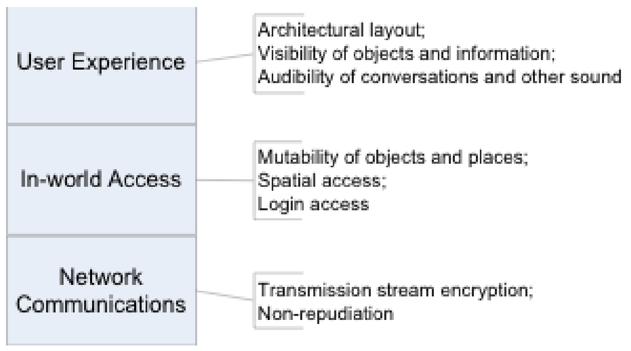
User Experience
Architectural layout;
Visibility of objects and information;
Audibility of conversations and other sound
In-world Access
Mutability of objects and places;
Spatial access;
Login access
Network Communications
Transmission stream encryption;
Non-repudiation

Fig. 1. Layers of security in a CVE

We believe that, within the in-world access layer, one way to bring about CVE privacy and integrity controls improvements is through a simplified, yet wide-ranging access control solution. Contrary to what is found in many commercial and open source CVEs, access control should be streamlined and ubiquitous so as to enable correct, consistent application across all virtual assets.

Our approach to securing the in-world access layer follows that of classic, UNIX-style, file system discretionary access control (DAC). We imitate this type of DAC primarily because it is well known in many general operating systems, easy to comprehend, and easy to deploy. We avoid more complex access control list systems, such as that found in Microsoft Windows XP/2000/2003, because their complexity can lead to misconfiguration [14].

The archetypal, UNIX-style DAC is based upon the assignment of permissions to three roles: the owner of a file system object, the group to which that object belongs, and all other system users. This basic DAC employs read, write, and execute permissions to permit or deny access to a directory or file; we propose something analogous though even simpler. The tenets of our system may be concisely stated as follows.

- Access controls must always be inherently simple to understand and manage.
- The CVE's virtual world should be viewed by access controls as a collection of objects: terrain, avatars, decor, accessories, media (i.e., sound, video, still image), etc.
- Access should be managed through two different permissions: *interact* and *alter*.
- All virtual world objects must have an assigned owner and participant group; a default *other* role includes all participants who are neither the owner nor in an object's group.
- *Interact* and *alter* permissions should be specified for an object's group and all other participants—the owner of an object will always have full access to the object.
- Any participant should be able to determine an object's ownership;[1] the owner of an object should be able to set that object's permissions, and may transfer ownership to a different participant.

Semantics of the *interact* and *alter* permissions are comparable to the classic read, write, and execute file system capabilities. *Interact* denotes read and execute, while *alter* is equivalent to write. In a VR context, interacting with an object entails viewing/hearing ("reading") and possibly using ("executing"), while altering an object involves changing, deleting, or updating in some way ("writing"). We considered splitting the *interact* permission into *observe* and *interact*, where *observe* would be equivalent to read and *interact* would be equivalent execute, but this seemed unnecessary: a participant should probably not interact with an object they cannot also observe, and if a participant is allowed to observe an object, there is probably an intention that they also have interact capabilities. Thus, collapsing *observe* and *interact* into just *interact* has merit for our purposes. Along with the roles of *owner*, *group*, and *other*, we believe the *interact* and *alter* permissions are sufficient to handle nearly all CVE access control situations. With any fewer permissions there is a significant loss of utility; more permissions, on the other hand, entail a risk of complexity, which may lead to improper access control configuration.

We have implemented a prototype of this DAC system, called WonderDAC, by extending Sun Microsystems' Project Wonderland version 0.3. We selected this CVE because of three important factors: first, Wonderland is at an early, formative stage, making it easy to integrate the components of our DAC system; next, Wonderland's approach to handling VR objects (called *cells* in Wonderland parlance) maps very well to the tenets we state above; and, finally, Wonderland's server component offers a logical, secure place to implement access controls [15].

The remainder of this paper is organized as follows: the next section summarizes related works; Section 3 presents background and details about Project Wonderland and its core architecture; Section 4 outlines critical use case scenarios the WonderDAC prototype should enable, Section 5 illustrates our prototype implementation; and Section 6 ends the paper with summary remarks and a brief discussion of further research and development.

II. RELATED WORK

As discussed in the previous section, we find there to be three basic layers in which CVE security controls should be implemented: network communications, in-world access, and user experience. Protecting the network communications layer has been a carefully studied problem in many other

[1] Unrestricted access to ownership information may raise privacy concerns. However, we believe the availability of owner IDs is in keeping with the classic, UNIX style of file system access control. Moreover, the benefit of making these IDs private is out- weighed by the cost of increased system complexity (potentially leading to confusion and access control misconfiguration).

research endeavors, the outcome of which is applicable to CVEs or any other systems that transmit data using network protocols. For example, among other techniques, Dit, Degrande, et al. [16], García, Montalà et al. [17], and Sato, Minamihata, et al. [18] all propose encrypting the network protocols that underlie their respective CVEs. Other network research peculiar to CVEs has touched on the integrity of the gaming experience. Along these lines, Baughman, Liberatore, et al. [19], Chiueh [20], and Neumann, Prigent, et al. [21] each consider how to provide consistent and fair participant interactions through methods such as dead-reckoning and bandwidth reduction.[2]

Regarding the middle CVE layer, in-world access, we find a limited number of efforts that deal with security controls. For example, Bullock and Benford [22] derive a spatial access control scheme, called SPACE, and note that through SPACE's adoption "a natural part of the environment is exploited, making it possible to hide explicit security mechanisms from end users through the natural spatial makeup of the environment." SPACE works by leveraging an access graph (a structure denoting constraints when moving from one location to another) to build adjacency and classification matrices. The adjacency matrix serves as a map, while the classification matrix determines the permissions a participant needs in order to move about a CVE. Pettifer and Marsh [23] propose a more refined, but complex, approach to access control, wherein CVE participants can interact with VR objects and exchange messages with one and other in order to gain access to resources. Central in this approach is the use of keys by participants to prove their identities and authority when undertaking some action.

Other means of in-world access control exploit VR metaphors. So-called privacy lamps and vampire mirrors are utilized by Butz, Beshers, et al. [24] to affect and verify object access. Privacy lamps shine a spotlight on virtual objects that are to be hidden from view, while vampire mirrors reveal objects that are private (such objects won't show up in the mirror) and can make an object private when a participant touches its mirror image.

Finally, some commercial, social networking CVEs offer in-world access controls, although these are typically geared toward the ownership, sale, and rental of virtual property. Second Life, for example, has extensive controls dealing with these issues, as do There, and Activeworlds [25-33]. The concept of role-based access (through a group) is also available in Second Life relative to land management. All three CVEs provide for other access control mechanisms that are tailored to the social networking experience (e.g., permissions to restrict flying and building). Often such controls require expertise with the CVE in question in order to be correctly applied and managed.

In the top CVE layer, user experience, consideration has been given to how structural divisions and social conventions play a role in participant behavior and privacy. Harrison and Dourish [34], and Honda, Tomioka, et al. [35, 36] explore how the design of virtual spaces can enable or discourage privacy by controlling views into work areas. The intersection of technical and social components in media environments (such as CVEs) is also examined by Mynatt, Adler, et al. [37], wherein it is noted that successful virtual communities must adapt and evolve practices and conventions from the real world. In so doing, visual cues (e.g., the layout of a space, signs) can affect participant behavior and the respect for others' virtual world privacy. Along these lines, Rissanen and Zheng [38] note that the privacy of a CVE participant (i.e., the right of that participant not to be hounded by other users in the CVE) may sometimes be an issue, too. Unfortunately, resetting a participant's virtual identity (their name and avatar appearance) may disrupt associations with friends or fail to work if the new identity becomes linked to the old one.

III. PROJECT WONDERLAND

3.1 Background

Project Wonderland is an open source, Java-based CVE that is constructed around a client-server model. Developed primarily by Sun Microsystems, Wonderland draws upon the technologies of four other open source projects (also operated by Sun):

- Project Darkstar: the server component for Wonderland; designed to offer a scalable, distributed, transaction-based, game server infrastructure
- jVoiceBridge: the audio server and client for Wonderland; provides spatial, stereo sound, as well as the ability to place telephone calls through the Session Initiation Protocol (SIP)
- Java 3D: responsible for providing the Wonderland client's 3D graphics and scene graph
- Project Looking Glass: used by the Wonderland client for 3D scene management (e.g., to create the client's user interface)

Compared to similar technologies, such as Croquet or Second Life, Wonderland is a relatively new offering in the realm of CVEs. First available in January 2007, this system incorporates a significant set of features, including, but not limited to: a scalable, virtual environment specified through a simple, XML file hierarchy; a transaction-based, fault-tolerant server that enables atomic participant activities; built-in avatar support (including a simple editor); spatial sound and audio chat; a virtual whiteboard; the ability to operate and share X Window applications within the Wonderland virtual world; and the means to connect a virtual phone with any SIP compliant phone system. The impetus for these features is a result of Sun's vision that Wonderland be "an environment that is robust enough in terms of security, scalability, reliability, and functionality that organizations can rely on it as a place to conduct real business [39]." These

[2] Dead-reckoning is a widely used method to improve the appearance of movement among participants and other objects in a virtual reality context. In essence, the position of a moving object is predicted based on its velocity and previous location. Updated information about the object's position is only required if it differs substantially from the prediction.

goals are meant to encompass remote office, educational, and other types of multi-user collaboration.

3.2 Client-Server versus Peer-to-Peer

Our ability to extend Wonderland with a DAC mechanism is enabled by Wonderland's client-server architecture. The client-server approach gives us a central authority, in the form of a server, that can be protected from malicious parties. By contrast, a peer-to-peer environment complicates the use of security controls, since each participant is on equal footing with full access to resources and information; serious problems can arise if one or more peers is under the control of a malicious user. As discussed by Neumann and Prigent [21], with regards to cheating in a peer-to-peer CVE-based game, numerous obstacles and issues must be resolved to enable reliable security controls:

> *First, it may be very complicated to apply cheating resistance extensively to a whole peer-to-peer architecture, that is mostly made of uncontrolled hosts. Moreover, it is unclear where to place the mechanisms to be used: on all the devices that take part to the game, or only on a subset of them? In this latter case how is this subset selected? Aside from this, in the case of a fully decentralized approach, the management of trust between the devices and between the players is a complex issue. How can we be sure that a device or a player acts faithfully? How is trust established and maintained? How is slandering handled?*

Endpoint trust, as one of the obstacles just touched upon, is an area of significant concern. In a client-server environment, clients must determine whether or not to trust servers and vice-versa; in a peer-to-peer environment, peers must determine whether or not to trust each other. Although we recognize the importance of endpoint trust in helping to establish access control, we believe it to be a matter that resides at the network communications layer, and, thus, is outside our stated scope of interest. Nevertheless, the area of public key infrastructure (PKI) offers well-studied solutions for client-server architectures. Peer-to-peer schemes also exist (e.g., [40] and [41]), but entail more complexity to compensate for the lack of a central authority.

3.3 Wonderland Cells and the Wonderland File System

Our implementation of the WonderDAC prototype takes place only in Wonderland's server components. However, as we will demonstrate in Section 5, there is still a need to enhance the client-side code in order to hide some of the cosmetic side-effects of WonderDAC. As mentioned above in Section 1, a Wonderland "...virtual world is composed of a collection of 'cells', each of which represents a 3D volume in the world [42]." A cell can denote a single decorative object in a virtual room, the room's architecture, or the entire virtual world. Hence, there is a hierarchical, parent-child structure at work in the use of cells, where a given cell may contain zero or more other cells. Moreover, this structure is represented by the XML files that prescribe the layout of a Wonderland virtual world and make up the Wonderland file system (WFS) [43]. Fig. 2 provides an example WFS hierarchy comprising a simplistic world. Each XML file is actually the serialization of a Java bean (called BasicCellGLOSetup) that handles the fundamentals of creating a generic cell. At the root of any WFS is a directory with a name ending in the -wfs suffix; within this directory XML files that end with a -wlc.xml suffix correspond to individual cells. It is possible to load up a cell file with an array of virtual object models: in this case, all objects are constituents of the cell. Alternatively, one can establish a parent-child relationship by creating a directory named after a given cell, but with a -wld suffix, and then populating that directory with new cell files (this is seen in Figure 2 where testRoomA is the parent of the cell given by Whiteboard-wlc.xml. All cells have virtual world coordinates that are relative to their parents' coordinates.

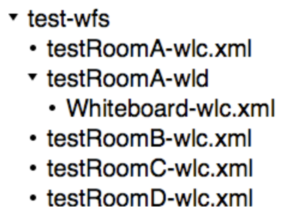
```
▾ test-wfs
  • testRoomA-wlc.xml
  ▾ testRoomA-wld
    • Whiteboard-wlc.xml
  • testRoomB-wlc.xml
  • testRoomC-wlc.xml
  • testRoomD-wlc.xml
```

Fig. 2. XML file hierarchy for a test Wonderland world

3.4 Communication

Two forms of communication are employed within the Wonderland client-server architecture: direct and publish/subscribe channels [42]. The former is primarily used in activities such as a client's initial connection to the Wonderland server, error message transmission, the management of shared applications, and handling virtual phone calls. The latter facilitates communications between clients and the server, as well as among clients (by way of the server). In the publish/subscribe channels, communications involve participants entering and leaving the Wonderland environment, cell management, avatar setup and movement, and server management (through a dedicated server management client).

The driving force behind using a combination of direct and publish/subscribe communications is efficiency: some situations call for the transmission of data between a given client and the server, while other situations may not involve the server in a logical sense. For example, if a participant enters or leaves the Wonderland environment, the server must be notified directly; if a participant causes a cosmetic change that only they and other nearby participants should see (e.g., a virtual object lights up when a mouse hovers over it) there is no need to update information on the server. In other words, there can be state changes on a client, and state changes on the server—figures 3 and 4 illustrate this. In Fig. 3, a server state change has taken place (step 1) as a result of some activity by Client A. In this circumstance, Client A directly contacts the server with an appropriate message (step 2); Client B is then updated by the server over a channel to which Client B subscribes (step 3). In Fig. 4, however, a cosmetic change has taken place within a cell that

Client A is accessing (step 1). Since there is no corresponding change in server state for the cell, the cell publishes an update to its channel (step 2a), which is subscribed to by Client B (step 2b).

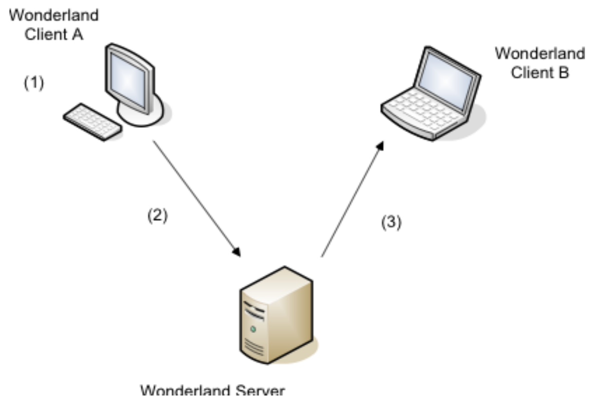

Fig. 3. Server state change: communications directly between a client and the server.

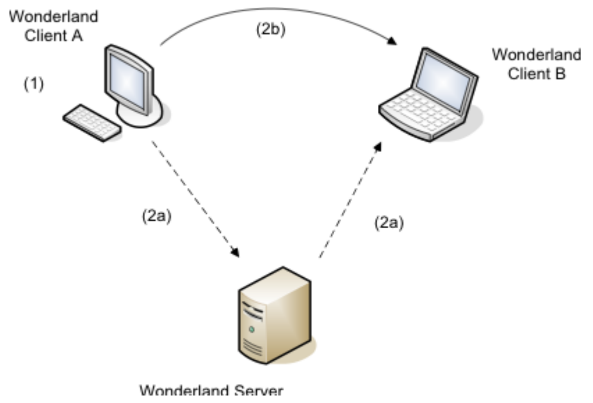

Fig. 4. Client-only state change: communications among Wonderland clients (by way of the server).

IV. USE CASE SCENARIOS

In Section 1, our list of tenets for the WonderDAC prototype started with the goal of simplicity and ease of understanding. We bear this in mind as we now consider five representative use case scenarios for CVE access control.

Our general approach with WonderDAC is modeled after the classic, UNIX-style DAC system, and, for a given VR object, works by assigning access permissions to three roles: *owner*, *group*, and *other*. As opposed to read, write, and execute, WonderDAC employs the VR analogs of *interact* (similar to read/execute) and *alter* (similar to write). By having the *interact* permission for an object, one can use, but not change, that object. By having the *alter* permission, one can change (i.e., update or delete) the object. As a participant operates within Wonderland, the privileges they are assigned for a given object are mutually exclusive: if the participant is a member of an object's group, then they will be assigned only the object's group permissions; permissions for an object's *other* role are assigned to a participant when the *group* role cannot be applied. This means that an object's *group* role can be used to grant or deny access to a select list of participants. It is important to bear in mind that the management of all WonderDAC permissions resides at the Wonderland server, as noted in Section 3.2. Aside from managing the issues of general access control and malicious clients, this configuration enables the owner of an object to efficiently assign their ownership to some other participant, avoiding some of the issues raised in [44]. Moreover, the centralization of access control affords us the ability to avoid problems such as deadlock when multiple participants attempt to utilize an object simultaneously (i.e., the Wonderland server becomes the system-of-record to sort things out).

Not all of the following use cases have been implemented in the WonderDAC prototype. Instead, we have focused on very basic components for use cases 1 and 2, laying the foundation for handling the remaining use cases in future work (see Section 6).

4.1 Use Case 1: Spatial Object Restrictions

A participant should only be able to enter, hear, and see into a virtual space if they are the owner of the space, a member of the space's group (and the group has *interact* permission), or the space is defined with *interact* permission for the *other* role. When a participant's avatar has entered a space, the avatar should only be visible and heard by other participants who also have *interact* permission (but may or may not be inside the space).

A participant should only be able to change a virtual space (by speaking in, or deleting/updating the space) if they are the owner of the space, a member of the space's group (and the group has *alter* permission), or the space is defined with *alter* permission for the *other* role. Although it is possible for a space to have only the *alter* permission assigned to the *group* or *other* roles, the result of such a configuration would be meaningless, since, to begin with, non-owners would be incapable of entering the space.

4.2 Use Case 2: Non-spatial Object Restrictions

A participant should only be able to view and, if applicable, hear a non-spatial object (e.g., a whiteboard, phone, or X Window application) if they are the owner of the object, a member of the object's group (and the group has *interact* permission), or the object is defined with *interact* permission for the *other* role. For example, a participant with only *interact* permission for a whiteboard can view the whiteboard and its contents, but cannot change or add to those contents. Similarly, a participant with only *interact* permission for a conference phone, can view and hear the phone, but cannot join in the conversation.

A participant should only be able to change/utilize a non-spatial object if they are the owner of the object, a member of the object's group (and the group has *alter* permission), or the object is defined with *alter* permission for the *other* role. Similar to Use Case 1, although it is possible for a non-spatial object to have only the *alter* permission assigned to the *group* and *other* roles, such a configuration is meaningless since non-owners would be unable to view/hear the object in the first place.

4.3 Use Case 3: Audio Conversation Restrictions

Two or more participants engaged in audio chat should be able to restrict their conversation to themselves (somewhat like having a "cone of silence" at their disposal). Other participants should be able to solicit the initial group to join in the conversation if the group wishes. This amounts to a specialized version of Use Case 1 except that an ephemeral, invisible space is created around the participants:

1. A temporary group role, to which all of the conversation participants belong, is created and assigned to the invisible space
2. *Interact* and *alter* permissions are assigned to the space's group role
3. New participants may be added to the temporary group role as desired by the initial conversation participants

It should be possible to assign/remove the *interact* and *alter* permissions for the invisible space's *other* role. This could enable, through the assignment of only the *interact* permission, members of the temporary group to converse, while all others can listen but not interrupt. The use of *alter* permissions alone for the invisible space's *other* role may or may not make sense, depending on the desires of the conversation participants: a participant in the *other* role would be able to speak in, but not hear, the conversation.

It is noteworthy that this use case addresses privacy and integrity concerns explored in other SIP/VR research. For example, Alam, Cohen, et al. [45] and Cohen [46] discuss the use of narrowcasting within SIP environments to implement granular restrictions on audio conference participants. Of particular interest is how Alam, Cohen, et al. employ a limited, Java3D-based CVE to configure, through narrowcasting, the speaking and hearing attributes of participants represented by avatars. Another example is offered by Benford and Fahlen [47] who leverage virtual analogs for the social components of in-person communication. Here, abstractions such as aura, awareness, and adapter are used to facilitate the ability of participants to speak with one and other. Similarity with this use case may be found in our employment of access-controlled space: somewhat akin to a combination of aura (a subspace that bounds participant interactions) and adapter (a mechanism that can modify aura and awareness) to protect a conversation. The means to control which conversants can hear and/or speak, and to whom they may listen and/or speak is an important tool—not only for security, but for improving the effectiveness of a conference with many participants.

4.4 Use Case 4: Avatar Cloaking

A participant should be able to disguise their avatar's appearance. Each avatar should have a *group* role that is unique to its owner. By assigning or removing *interact* permissions to/from this group or to/from the avatar's *other* role, the appearance of the avatar should be controllable by its owner. A participant should be able to see an avatar's true image if they are a member of the avatar's group (and the group has *interact* permission), or the avatar is defined with *interact* permission for the *other* role. Otherwise, a generic image should be displayed by the avatar. The use of *alter* permissions should be ignored, here: only the owner should be permitted to make changes to an avatar's appearance. This approach helps to manage any risk of avatar defacement by deferring to the owner to facilitate all avatar image changes.

4.5 Use Case 5: Permissions and Ownership Changes

All participants should be able to determine the ownership of a VR object. Also, a participant should be able to change permissions for the *group* and *other* roles of any object they own, or, if desired, assign ownership of the object to another participant. The interface to carry out these activities should be simple and accessible through mouse interactions with the object in question.

4.6 Summary Use case Table

Table 1, at the end of this article, presents a summary of the use case scenarios discussed above, along with the semantics behind permissions settings.

V. PROTOTYPE IMPLEMENTATION

As discussed in Section 4, our focus during the implementation of this prototype was on very basic components for use cases 1 and 2. To begin with, we leveraged Wonderland's ability to use the lightweight directory access protocol (LDAP) as the means of authenticating a participant.[3] We extended this to include authorization by storing a group membership list in each participant's LDAP record. When a participant logged into Wonderland to authenticate, their membership list was accessed and parsed into an array of group names, which was then stored in an associated WonderlandIdentity object. Due to constraints in Darkstar, we had to implement a special Darkstar service to enable access to a participant's identity information during the login process.

Next, we took advantage of a pre-existing, empty, class stub that Wonderland developers had inserted into their source code as a placeholder for future access control checks. We replaced this stub (called CellAccessControl) with a new class that could compare a given cell's group with the group membership of a participant, and then, based on the outcome, perform subsequent permissions checks. This new class put into effect the mutually exclusive assignment of permissions discussed at the outset of Section 4: "...if the participant is a member of an object's group, then they will be assigned only the object's group permissions; permissions for an object's *other* role are assigned to a participant when the *group* role cannot be applied."

Finally, we made modifications to other central Wonderland classes. This included, but was not limited to: CellGLO, the parent class for all Wonderland cells; BasicCellGLOSetup, a utility class to aid in configuring a cell; and WhiteboardCellGLO, the server component of a Wonderland whiteboard cell. Our modifications to

[3] This ability also includes support for strongly encrypted communications between the LDAP and Wonderland servers.

BasicCellGLOSetup enabled us to add the following property tags to the XML files that specify Wonderland cells:

- accessOwner – The name of the cell's owner
- accessGroup – The name of the cell's group
- accessGroupPermissions – A number indicating the *interact* and *alter* permissions assigned to the cell's group ("IA" = 3, "I-" = 2, "-A" = 1, "--" = 0)
- accessOtherPermissions – A number indicating the *interact* and *alter* permissions assigned to the cell's *other* role.

5.1 Spatial Object Access

Here, we demonstrate how the WonderDAC prototype was able to control spatial access. For this situation, there were two cells (spaces) of interest: a two-story room referred to as testRoomC and, within this space, a second story loft with stairs leading up, referred to as testRoomD. For the purposes of this demonstration, testRoomC was configured as follows: (twright, admin, 2, 2). This tuple is shorthand for denoting that the cell's owner is *twright*, the cell's group is *admin*, the group permissions are *I-*, and the permissions for the *other* role are *I-*. The loft, testRoomD, was configured with: (twright, admin, 0, 0), where a 0 denotes no permissions granted. In Fig. 5, we see two avatars, twright and bench-40, looking at the spaces created by testRoomC and testRoomD; this figure is from twright's perspective. Because twright is the owner of both cells, he is able to see both rooms. Fig. 6, however, is from bench-40's perspective, and does not include the presence of testRoomD. This is because bench-40 is a guest user who does not belong to the admin group, and testRoomD permits neither *interact* nor *alter* access for its group or the *other* roles.

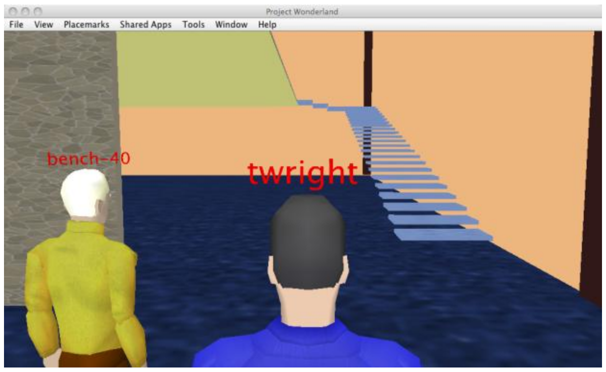

Fig. 5. The avatar twright looks on at testRoomD and testRoomC.

Although the bench-40 participant is unable to view testRoomD, our current WonderDAC prototype stops short of concealing from them the avatars that enter testRoomD. Such concealment is required to fully implement Use Case 1, and is planned future work. Fig. 7 depicts what happens when twright enters into the testRoomD cell as bench-40 (an avatar that cannot enter this space) watches.

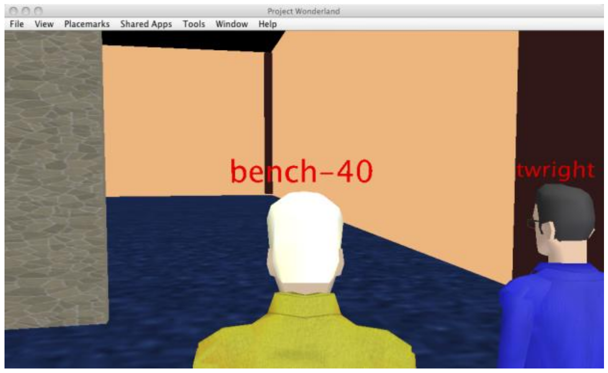

Fig. 6. The avatar bench-40 looks on at testRoomC, but cannot see testRoomD.

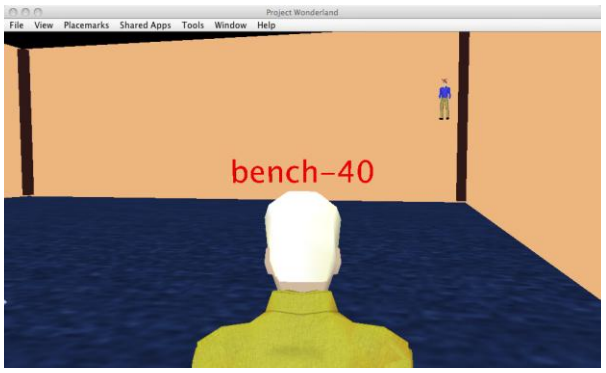

Fig. 7. The avatar bench-40 looks in the direction of testRoomD as twright climbs the stairs.

5.2 Non-spatial Object Access

To demonstrate non-spatial object access, we created a whiteboard cell in our test Wonderland environment and configured it in the same manner as testRoomC: (twright, admin, 2, 2). Both the twright and bench-40 participants were able to see the whiteboard object, including images drawn there. However, only twright was supposed to be able to change the contents, since, as the owner, twright was the only participant with *alter* permissions for the whiteboard object. This proved to be an interesting situation for Wonderland due to its use of publish/subscribe channels. In particular, when a participant uses a whiteboard, they are operating on a local copy maintained by their Wonderland client. Periodically, their client publishes updates of the whiteboard's state so that other clients observing/participating with the whiteboard may stay synchronized. If a participant lacks *interact* permission for the whiteboard they will, of course, be unable to see or use it in any way. If they lack just *alter* permission, they will still be able to draw on their local copy, even though the Wonderland server will ignore any whiteboard updates their client tries to send. Hence, there is a cosmetic issue: even though no other participants will see an unauthorized participant's whiteboard updates, the unauthorized participant can continue to use their whiteboard locally. Figs. 8 and 9 illustrate this happening with twright and bench-40.

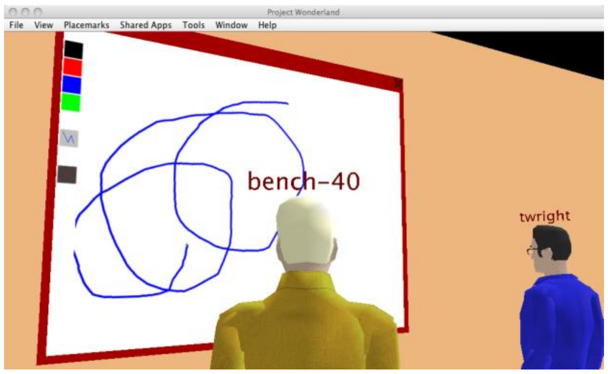

Fig. 8. The bench-40 participant draws on a local copy of the whiteboard shared with twright.

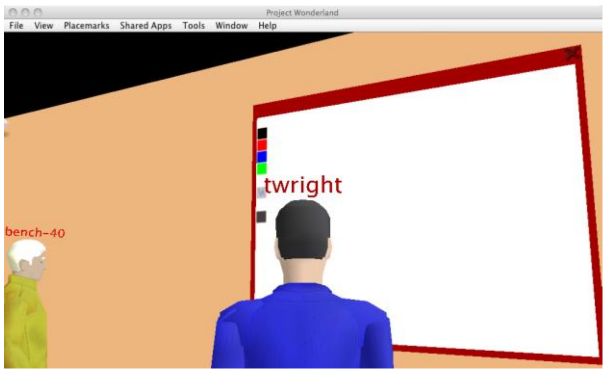

Fig. 9. Because bench-40 has no alter permission, the twright participant cannot see bench-40's whiteboard updates.

As an authorized participant uses a whiteboard, their updates will be added to the local copies of all participants—some of whom may not have alter permission, but have drawn on their local whiteboards nonetheless. The place to fix this issue is at the Wonderland client. Because there is no security risk involved (just the nuisance of a cluttered whiteboard for some participants), enhancing the client to block whiteboard use when a participant has no *alter* permission should be a reasonable solution. A complete implementation of Use Case 2 requires such an enhancement, and is planned for future work. We do not believe a similar circumstance exists for Wonderland's virtual phone.

VI. CONCLUSION AND FUTURE WORK

As CVEs continue their evolution into more ubiquitous, useful technologies, we find them enabling entertainment, commercial, educational, and scientific endeavors. A critical component to the success of any collaborative technology is the control of access to information and resources. For most CVEs (commercial and open source) such control is often limited in scope and not designed to fully leverage the assignment of roles to a participant. In answer to this, we have implemented a limited prototype called WonderDAC: a discretionary access control system for the Project Wonderland CVE. We have designed WonderDAC not only to be an effective means of access control, but also to serve as a simple, ubiquitous mechanism that may be consistently applied by CVE participants. We believe this contrasts with other CVEs (both commercial and open source) where discretionary access controls may be narrowly focused, complex, and/or significantly lacking in capabilities.

WonderDAC is modeled after a classic, UNIX-style DAC system, and treats all virtual objects according to three roles: *owner*, *group*, and *other*. *Interact* and *alter* permissions are assigned to, or removed from the *group* and *other* roles to control access by CVE participants, while an object's owner always maintains full privileges. The *interact* permission is analogous to read/execute in a UNIX-style DAC system, while *alter* is similar to write. By defining five representative use case scenarios for managing CVE access control, we formulated a guide for our prototype implementation.

Although our prototype is limited to just the first two use cases (access control for spatial and non-spatial objects), it provides an important foundation for future work. We have undertaken additional re-search to expand upon WonderDAC, including short-term and long-term objectives. The short-term work deals with cosmetic issues raised by the prototype: hiding avatars that reside in privileged spaces from the view of unauthorized participants, and modifying the Wonderland client to prevent unauthorized changes to a participant's local whiteboard and other virtual objects. The long-term work addresses the remaining use case scenarios, entails building an interface for participants to view and manage object ownership and permissions information, and includes a realistic demonstration of WonderDAC with human participants. Other topics that we further expand upon include network communications security and whether or not there is a need for a traditional, UNIX-style *root* account (in theory, such an account should not be required by WonderDAC, although temporary limitations in Wonderland may make an administrator role necessary under some circumstances).

REFERENCES

[1] J. E. Bendis, Developing educational virtual worlds with game engines, in SIGGRAPH '07: *ACM SIGGRAPH 2007 educators program*. 2007, ACM: New York, NY, USA. pp. 26.
[2] A. Le Blanc, J. Bunt, J. Petch, and Y. Kwok, The virtual learning space: an interactive 3D environment, in Web3D '05: *Proceedings of the tenth international conference on 3D Web technology*. 2005, ACM: New York, NY, USA. pp. 93--102.
[3] D. A. Bray and B. R. Konsynski, Virtual worlds: multi-disciplinary research opportunities. *SIGMIS Database*, 2007. 38(4): p. 17--25.
[4] T. E. Wright and G. Madey, A Survey of Technologies for Building Collaborative Virtual Environments. *International Journal of Virtual Reality*, 2009. 8(1): pp. 53--66.
[5] D. Terdiman. Newsmaker: *Virtual magnate shares secrets of success*. 2006 June 6, 2008; Available from: http://news.cnet.com/2102-1043_3-6144967.html.
[6] J. Brownlee. *John Edwards Meets Second Life 'Feces Spewing Obscenity*'. 2007 June 6, 2008; Available from: http://blog.wired.com/tableofmalcontents/2007/03/john_edwards_me.html.
[7] M. Wagner. Toyota Under Attack By Second Life Griefers. 2007 June 6, 2008; Available from: http://www.informationweek.com/blog/main/archives/2007/04/toyota_under_at.html.
[8] J. Dibbell. Mutilated Furries, Flying Phalluses: Put the Blame on Griefers, the Sociopaths of the Virtual World. 2008 June 6, 2008;

Available from: http://www.wired.com/print/gaming/virtualworlds/magazine/16-02/mf_goons.

[9] A. Fass. OutFront: Sex, Pranks and Reality. 2007 June 6, 2008; Available from: http://www.forbes.com/home/free_forbes/2007/0702/048.html.

[10] Linden Research, Inc. Second Life Viewer Susceptible to Quicktime Security Flaw. 2007 June 6, 2008; Available from: http://blog.secondlife.com/2007/11/30/second-life-viewer-susceptible-to-quicktime-security-flaw/.

[11] A. B. Brown, Oops! Coping with human error in IT systems. Queue, 2004. 2(8): pp. 34--41.

[12] R. Scandariato, B. D. Win, and W. Joosen, Towards a measuring framework for security properties of software, in QoP '06: Proceedings of the 2nd ACM workshop on Quality of protection. 2006, ACM: New York, NY, USA. pp. 27--30.

[13] P. Schmehl, Barbarians at the gateway, defeating viruses in EDU, in SIGUCCS '01: Proceedings of the 29th annual ACM SIGUCCS conference on User services. 2001, ACM: New York, NY, USA. pp. 177--180.

[14] S. Govindavajhala and A. W. Appel, Windows Access Control Demystified. 2006, Department of Computer Science, Princeton University.

[15] J. C. Weber and T. Parisi, An open protocol for wide-area multi-user X3D, in Web3D '07: Proceedings of the twelfth international conference on 3D web technology. 2007, ACM: New York, NY, USA. pp. 133--136.

[16] S. L. Dit, S. Degrande, C. Gransart, C. Chaillou, and G. Saugis, VRML97 distributed authoring interface, in Web3D '03: Proceeding of the eighth international conference on 3D Web technology. 2003, ACM: New York, NY, USA. p. 135--ff.

[17] P. Garc ía, O. Montal à, C. Pairot, R. Rallo, and A. G. Skarmeta, MOVE:: component groupware foundations for collaborative virtual environments, in CVE '02: Proceedings of the 4th international conference on Collaborative virtual environments. 2002, ACM: New York, NY, USA. pp. 55--62.

[18] F. Sato, K. Minamihata, and T. Mizuno, A totally ordered and secure multicast protocol for distributed virtual environment. Parallel and Distributed Systems: Workshops, Seventh International Conference on, 2000, Oct 2000: pp. 55-60.

[19] N. E. Baughman, M. Liberatore, and B. N. Levine, Cheat-proof playout for centralized and peer-to-peer gaming. IEEE/ACM Trans. Netw., 2007. 15(1): pp. 1--13.

[20] T.-C. Chiueh, Distributed systems support for networked games. Operating Systems, 1997., The Sixth Workshop on Hot Topics in, 1997: pp. 99-104.

[21] C. Neumann, N. Prigent, M. Varvello, and K. Suh, Challenges in peer-to-peer gaming. SIGCOMM Comput. Commun. Rev., 2007. 37(1): pp. 79--82.

[22] A. Bullock and S. Benford, An access control framework for multi-user collaborative environments, in GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work. 1999, ACM: New York, NY, USA. pp. 140--149.

[23] S. Pettifer and J. Marsh, Collaborative access model for shared virtual environments. Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001. WET ICE 2001. Proceedings. Tenth IEEE International Workshops on, 2001: pp. 257-262.

[24] A. Butz, C. Beshers, and S. Feiner, Of Vampire mirrors and privacy lamps: privacy management in multi-user augmented environments, in UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology. 1998, ACM: New York, NY, USA. pp. 171--172.

[25] Activeworlds, Inc. Online Help Manual: World Admin Rights Dialog. 2008 March 23, 2008; Available from: http://www.activeworlds.com/help/aw41.

[26] Activeworlds, Inc. Online Help Manual: World Admin Features Dialog. 2008 March 23, 2008; Available from: http://www.activeworlds.com/help/aw41.

[27] Activeworlds, Inc. Online Help Manual: Shared Privileges. 2008 March 23, 2008; Available from: http://www.activeworlds.com/help/aw41.

[28] Linden Research, Inc. Knowledge Base: Permissions. 2007 March 4, 2008; Available from: https://support.secondlife.com/ics/support/security.asp.

[29] Linden Research, Inc. Knowledge Base: Land and the Linden Dollar Economy. 2008 March 4, 2008; Available from: https://support.secondlife.com/ics/support/security.asp.

[30] Linden Research, Inc. Knowledge Base: Inworld Issue (cooperative object editing). 2007 March 4, 2008; Available from: https://support.secondlife.com/ics/support/security.asp.

[31] Makena Technologies, Inc. Getting Around: Understanding Permissions. 2004 March 23, 2008; Available from: http://info.there.com/article.php?id=105.

[32] Makena Technologies, Inc. Development Lot Setup Manual. 2008 March 23, 2008; Available from: http://info.there.com/article.php?id=1681.

[33] Makena Technologies, Inc. Giving and Receiving. 2006 March 23, 2008; Available from: http://info.there.com/article.php?id=076.

[34] S. Harrison and P. Dourish, Re-place-ing space: the roles of place and space in collaborative systems, in CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work. 1996, ACM: New York, NY, USA. pp. 67--76.

[35] S. Honda, H. Tomioka, T. Kimura, T. Ohsawa, K. Okada, and Y. Matsushita, A virtual office environment based on a shared room realizing awareness space and transmitting awareness information, in UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology. 1997, ACM: New York, NY, USA. pp. 199--207.

[36] S. Honda, H. Tomioka, T. Kimura, T. Ohsama, K. Okada, and Y. Matsushita, A virtual office environment for home office worker based on 3D virtual space. Virtual Systems and MultiMedia, 1997. VSMM '97. Proceedings., International Conference on, 10-12 Sep 1997: pp. 38-47.

[37] E. D. Mynatt, A. Adler, M. Ito, and V. L. O'Day, Design for network communities, in CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems. 1997, ACM: New York, NY, USA. pp. 210--217.

[38] M. J. Rissanen and X. Zheng, On privacy of famous users in active worlds. SICE 2004 Annual Conference, 2004. 1: pp. 35-38 vol. 1.

[39] Sun Microsystems. Project Wonderland Web Site. 2008 February 13, 2008; Available from: http://lg3d-wonderland.dev.java.net.

[40] K. Berket, A. Essiari, and A. Muratas, PKI-based security for peer-to-peer information sharing. Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, 2004: pp. 45-52.

[41] C. Sturm, K. R. Dittrich, and P. Ziegler, An access control mechanism for P2P collaborations, in DaMaP '08: Proceedings of the 2008 international workshop on Data management in peer-to-peer systems. 2008, ACM: New York, NY, USA. pp. 51--58.

[42] Sun Microsystems. Project Wonderland Software Architecture. 2008 February 14, 2008; Available from: http://wiki.java.net/bin/view/Javadesktop/ProjectWonderlandArchitecture.

[43] Sun Microsystems. Tutorial: How to Create a World Using a Wonderland Filesystem (WFS). 2008 March 27, 2008; Available from: http://wiki.java.net/bin/view/Javadesktop/ProjectWonderland WFS.

[44] M. Oliveira, J. Jordan, J. Pereira, J. Jorge, and A. Steed, Analysis Domain Model for Shared Virtual Environments. The Journal of Virtual Reality, 2009. 8(4): pp. 1--30.

[45] M. S. Alam, M. Cohen, J. Villegas, and A. Ahmed, Figurative Privacy Control of SIP-Based Narrowcasting. Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on, 2008: pp. 726-733.

[46] M. Cohen, Exclude and Include for Audio Sources and Sinks: Analogs of Mute \& Solo Are Deafen \& Attend. Presence: Teleoperators \& Virtual Environments, 2000. 9(1): pp. 84-96.

[47] S. Benford and L. Fahl\'en, A spatial model of interaction in large virtual environments, in ECSCW'93: Proceedings of the third conference on European Conference on Computer-Supported Cooperative Work. 1993, Kluwer Academic Publishers: Norwell, MA, USA. pp. 109--124.

TABLE 1: SUMMARY OF ACCESS CONTROL USE CASES

Use Case	Permissions	Meaning for Group Role	Meaning for Other Role
Spatial Object	IA	Any member of the object's group can enter, see, and hear into the space. All such constituents can speak within and change the space, too.	All participants who are not members of the object's group can enter, see, and hear into the space. All such constituents can speak within and change the space, too.
	I-	Any member of the object's group can enter, see, and hear into the space. All such constituents are unable to speak within and change the space.	All participants who are not members of the object's group can enter, see, and hear into the space. All such constituents are unable to speak within and change the space.
	-A	Meaningless: a member of the object's group can speak within and change the space, but they are unable to enter the space.	Meaningless: all participants who are not members of the object's group can speak within and change the space, but they are unable to enter the space.
	--	No member of the object's group can enter, see, or hear into the space. All such constituents are unable to speak within and change the space.	All participants who are not members of the object's group cannot enter, see, or hear into the space. All such constituents are unable to speak within and change the space.
Non-spatial Object	IA	Any member of the object's group can see and, if applicable, hear the object. All such constituents can change the object, too.	All participants who are not members of the object's group can see and, if applicable, hear the object. All such constituents can change the object, too.
	I-	Any member of the object's group can see and, if applicable, hear the object. All such constituents are unable to change the object.	All participants who are not members of the object's group can see and, if applicable, hear the object. All such constituents are unable to change the object.
	-A	Meaningless: a member of the object's group can change the object, but they are unable to see or, if applicable, hear the object.	Meaningless: all participants who are not members of the object's group can change the object, but they are unable to see or, if applicable, hear the object.
	--	No member of the object's group can see, change, or, if applicable, hear the object.	All participants who are not members of the object's group cannot enter, see, change, or, if applicable, hear the object.
Audio Conversation	IA	All member of the conversation's temporary group can hear and speak in the conversation.	All participants who are not members of the conversation's temporary group can hear and speak in the conversation.
	I-	Not allowed: all members of the conversation's temporary group can hear the conversation, but not speak. This would prevent the conversation from taking place.	All participants who are not members of the conversation's temporary group can hear, but not speak in, the conversation.
	-A	Not allowed: all members of the conversation's temporary group can speak in, but not hear the conversation. This would prevent the conversation from taking place.	All participants who are not members of the conversation's temporary group can speak in, but not hear, the conversation. This may or may not be useful, contingent on the desires of the conversation's temporary group members.
	--	Not allowed: no member of the conversation's temporary group can speak in or hear the conversation. This would prevent the conversation from taking place	All participants who are not members of the object's group cannot speak in or hear the conversation.
Avatar Cloak	IA	Any member of the object's group can see the true appearance of the avatar. The *alter* permission is ignored: only the owner is allowed to change an avatar.	All participants who are not members of the avatar's group can see the true appearance of the avatar. The *alter* permission is ignored: only the owner is allowed to change an avatar.
	I-	Any member of the object's group can see the true appearance of the avatar. (Effectively the same as *IA* permissions above)	All participants who are not members of the avatar's group can see the true appearance of the avatar. (Effectively the same as *I A* permissions above)
	-A	No member of the avatar's group sees the avatar's true image; instead, they see a generic disguise. The *alter* permission is ignored: only the owner is allowed to change an avatar.	All participants who are not members of the avatar's group cannot see the avatar's true image; instead, they see a generic disguise. The *alter* permission is ignored: only the owner is allowed to change an avatar.
	--	No member of the avatar's group sees the avatar's true image; instead, they see a generic disguise. (Effectively the same as *-A* permissions above)	All participants who are not members of the avatar's group cannot see the avatar's true image; instead, they see a generic disguise. (Effectively the same as *-A* permissions above)

Timothy E. Wright, Ph.D. Dr. Wright is a computational scientist working at the University of Notre Dame's Center for Research Computing. His research interests include virtual reality and information security. In the past he has worked as a software engineer, computer fraud and abuse investigator, information security professional, and senior IT auditor. Dr. Wright is also a member of the ACM, ISACA, and ISC^2.

Gregory Madey, Ph.D. Dr. Madey's research focuses on the use of computer science to develop solutions to a wide range of problems. Recent problem domains include environomental science, the open source software phenomenon, decision support for cellular phone networks, mobile ad hoc sensor networks, and bioinformatics. His research is interdisciplinary and draws on theories and topics from database theory, e-Technologies, agent-based simulation, artificial intelligence, emergence and self-organization, chaos and complexity, organizational theory, artificial neural networks, data mining, and operation research.