University of California Curation Center

GhOST: Merritt Access Control

Rev. 0.5 – 2011-05-04

1 Introduction

Information technology and resources have become integral and indispensable to the pedagogic mission of the University of California. Members of the UC community routinely produce and utilize a wide variety of digital assets in the course of teaching, learning, and research. These assets represent the intellectual capital of the University; they have inherent enduring value and need to be managed carefully to ensure that they will remain available for use by future scholars. Within the UC system the UC Curation Center (UC3) of the California Digital Library (CDL) has a broad mandate to ensure the long-term usability of the digital assets of the University. UC3 views its mission in terms of *digital curation*, the set of policies and practices aimed at maintaining and adding value to authentic digital assets for use by scholars now and into the indefinite future [Abbott].

In order to meet these obligations UC3 is developing Merritt, an emergent approach to digital curation infrastructure [Merritt]. Merritt devolves infrastructure function into a growing set of granular, orthogonal, but interoperable micro-services embodying curation values and strategies [Foundations]. Since each of the services is small and self-contained, they are collectively easier to develop, deploy, maintain and enhance [Denning]; equally as important, since the level of investment in and commitment to any given service is small, they are more easily replaced when they have outlived their usefulness. Yet at the same time, complex curation functionality can emerge from the strategic combination of individual, atomistic services [Fisher].

GhOST ("GO-or-STop") is a micro-service providing granular access control for digital content managed in a Merritt curation environment. GhOST centrally manages information of use to client services in making and enforcing local access decisions.

NOTE The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [RFC2119].

NOTE Italics are used to indicate sets, e.g., *AP*, and set members or variables, e.g., $ap \in AP$. A subscripted variable indicate a particular, but unspecified value, e.g., ap_0. In other words, a subscripted variable is a *bound* variable while an unsubscripted one is a *free* variable. Quotation marks indicate specific literal values, e.g., 'mrt:any'.

NOTE The initial implementation of GhOST makes two simplifying assumptions: (1) all role/context-based authorization rules are permissive in nature, that is, they explicitly specify what access *is* allowed (as opposed to being prohibitionary and specifying what access is *not* allowed); and, perhaps more significantly, (2) for purposes of authorization all information resources are associated with a *single* context. The relaxation of these constraints may be dealt with in subsequent versions of the service.

2 Access Control

Access control encompasses a number of individual functions supportive of an enforceable *decision*, $de \in \{\text{true, false}\}$, regarding whether or not a *user agent*, $ua \in UA$, with an authenticated identity, $id \in ID$, can perform an *operation*, $op \in OP$, on an *information resource*, $ir \in IR$, in a service *application*, $ap \in AP$. This can be expressed in set-theoretic and functional notation as:

$$\text{Authz: } ID \times OP \times IR \times AP \rightarrow \{\text{'true', 'false'}\}$$
$$de \leftarrow \text{authz}(id, op, ir, ap)$$

A comprehensive access control solution needs to support at least three key functions:

- *Identity management* (IdM) – Management of user accounts and pertinent characteristics.
- *Authentication* (authn) – Verification of an assertion of identity, or in other words, is a user agent who he/she/it purports to be?
- *Authorization* (authz) – Determination of permission, or in other words, is an authenticated user agent permitted to perform a given operation on a given resoruce?

2.1 Role/Context-Based Authorization

It is generally combinatorially infeasible to maintain extensive authorization rules on a per-identity, per-resource basis. However, this can be made tractable if access decisions are made relative to *aggregate*, rather than *individual*, characteristics of agents and resources, conventionally known as *roles*, $ro \in RO$, for user agents and *contexts*, $co \in CO$, for resources (under the assumption that $\|RO\| << \|ID\|$ and $\|CO\| << \|IR\|$).

$$\text{Authz: } RO \times OP \times CO \times AP \rightarrow \{\text{'true', 'false'}\}$$
$$de \leftarrow \text{authz}(ro, op, co, ap)$$

An authorization *rule* is a tuple, $ru = \{ro, op, co, ap, de\}$. Conceptually, the set of all rules, RU, can be interpreted as an associative array with a composite role/operation/context/application key and boolean decision value:

Role	***Operation***	***Context***	***Application***	***Decision***
ro	*op*	*co*	*ap*	*de*
…				

(For illustrative purposes this is represented in tabular form; however, this does not necessarily imply an RDBMS implementation.)

NOTE For complete generality, authorization rules may be defined in either a permissive (de = 'true') or prohibitionary (de = 'false') sense. As a pragmatic optimization, GhOST will initially only support permissive rules. Unless an operation and resource are explicitly addressed by a permissive rule, that operation is prohibited. Nevertheless, the decision element (de = 'true') is explicitly expressed in all tuples to facilitate future support of prohibitionary rules.

The authorization function can return a *ruleset*, $RS = \{ru_i, \ldots\}$ by leaving at least one of its parameters as a free, rather than bound variable. For example, authz(op_0, co_0, ap_0) returns the set of all rules with arbitrary role but specific operation, context, and application.

$$ru_i = \{ro, op, co, ap, de\} \mid op = op_0 \wedge co = co_0 \wedge ap = ap_0$$

Roles and contexts are opaque values whose semantics need be understood only by the relevant application. Any values easily derivable for identities and resources can be used for this purpose. However, they should be selected carefully as the level of granular access control cannot be finer than the sets of identities and resources sharing the same role and context ($RO \times CO$).

The reserved wildcard token "mrt:any" indicates *any* role, operation, context, or application in an authorization rule. Thus, the ruleset:

Role	**Operation**	**Context**	**Application**	**Decision**
mrt:any	mrt:any	mrt:any	ap_0	true
curator	mrt:any	co_1	ap_1	true
admin	mrt:any	mrt:any	mrt:any	true

permits unfettered public access to all operations on all ap_0 resources; access to all operations on ap_1 resources with the co_1 context to agents holding the "curator" role; and access to all operations on all resources in all applications to agents holding the "administrator" role.

Resource-to-context mappings are best managed directly by the applications in which the resources themselves are managed, rather than being dealt with centrally by GhOST, as the applications are better able to apply appropriately-nuanced distinctions. (For example, the Merritt repository already implicitly knows of which collections a given object is a member, and EZID implicitly knows to which owner group a given identifier is associated.) An application MUST be capable of mapping every resource managed in it to a GhOST context.

$$\text{Context}_{\text{ap}}: IR \rightarrow CO$$
$$co \leftarrow \text{context}_{\text{ap}}(ir)$$

User agent-to-role mappings are best managed centrally by GhOST.

2.2 User Agent Profile

User agent roles may be dependent on the agent's *profile*, $pr \in PR$, a set of typed descriptive *characteristics*, $ch \in CH$. These types can be generic, $tg \in TG$ (e.g., agent title, contact information, institutional affiliation, etc.), application-specific, $ta \in TA$ (e.g., UI preferences, etc.), or application/context-specific, $tc \in TC$ (e.g., agent role, etc.).

$$
\begin{aligned}
&\text{Profile: } ID \times TG \rightarrow CH \\
&\text{Profile: } ID \times TA \times AP \rightarrow CH \\
&\text{Profile: } ID \times TC \times AP \times CO \rightarrow CH \\
&ch \leftarrow \text{profile}(id, tg) \\
&ch \leftarrow \text{profile}(id, ta, ap) \\
&ch \leftarrow \text{profile}(id, tc, ap, co)
\end{aligned}
$$

Note that the profile and authorization functions share a common set of applications, *AP*, and contexts, *CO*.

Generic types include:

- Agent name.
- Agent type: person, pseudo-person, corporate, automated.
- Agent affiliation.
- Agent address.
- Agent telephone.
- Agent email.
- Time zone preference.
- Public/private authentication credentials (e.g., username/password).

$TG = \{$'name', 'type', 'affiliation', 'address', 'telephone', 'email', 'timezone', 'username', 'password', …$\}$

Application-specific types include:

- *TBD*.

Application/context-specific types include:

- Agent role.

$TC = \{$'role', …$\}$

$ro \leftarrow \text{profile}(id, \text{'role'}, ap, co)$

The reserved token 'mrt:any' may be used for application and context wildcarding. The former is appropriate for defining cross-application administrative roles, while the latter is appropriate for defining cross-context curatorial roles.

All of an agent's roles in a given application can be collected into a *roleset*, $RT = \{ro_i, \ldots\}$.

2.3 Authentication

A user agent can be an individual person (or pseudo-person) or an automated system. Individual identities are authenticated with public/private credentials, for example, username, $un \in UN$, and password, $pw \in PW$, or by an IP address; automated systems are authenticated by IP address.

$$\text{Authn: } UN \times PW \rightarrow ID$$
$$\text{Authn: } IP \rightarrow ID$$
$$id \leftarrow \text{authn}(un, pw)$$
$$id \leftarrow \text{authn}(ip)$$

2.3.1 Single Sign-On (SSO) Authentication Session

To avoid the necessity for repeated authentication challenges, GhOST supports the notion of an authentication *session*. GhOST establishes a session *state* ($st \in ST$) that is associated with all subsequent user agent requests up to an expiration time, encapsulating the following information:

- GhOST version identifier
- GhOST instance identifier
- Authenticated identity
- Authentication scheme: "challenge" or "ip"
- Authentication date/time
- Expiration date/time

$$st = \{ve, in, id, sc, au, ex\}$$

The authentication scheme indicates the basis of the authentication process, 'challenge" indicating verification of the user agent's public/private credentials; and 'ip' indicating verification of an IP address in a known range.

GhOST will ultimately support UCTrust/Shibboleth-based authentication using local campus identity providers for UC-affiliated user agents.

3 Access Control Workflow

The general processing of a request by user agent ua_0 for operation op_0 on resource ir_0 in application ap_0 can be summarized as follows:

1. Retrieve the resource context, $co_0 \leftarrow \text{context}_{ap}(ir_0)$. Note that this is the responsibility of the

application ap_0 based on its local knowledge of the resource ir_0.

2. Retrieve the ruleset associated with the specified operation (or wildcard) and context (or wildcard), $RS_0 \leftarrow$ authz($op_0 \cup$ ‘mrt:any’, $co_0 \cup$ ‘mrt:any’, ap_0).
3. If a rule with a wildcard role and a permissive decision does not exist, $\neg\exists\ ru_i \in RS_0 \mid ru_i =$ {‘mrt:any’, op_0, co_0, ap_0, true}, in which case unauthenticated and unauthorized access is not allowed, then:
    - 3.1. If at least one rule with a permissive decision does not exist, $\neg\exists\ ru_i \in RS_0 \mid ru_i = \{ro_0, op_0, co_0, ap_0, \text{true}\}$, in which case authenticated and authorized access is not allowed, then:
        - 3.1.1. Return an appropriately-formatted error response.
        - 3.1.2. Terminate the processing of the request.
    - 3.2. If at least one rule with a permissive decision does exist, $\exists\ ru_i \in RS_0 \mid ru_i = \{ro_0, op_0, co_0, ap_0, \text{true}\}$, in which case authenticated and authorized access is allowed, then:
        - 3.2.1. If an unexpired session state does not exist, $\neg\exists\ st_0 = \{\ldots, id_0, \ldots, ex_0\} \mid ex_0 < now$, then:
            - 3.2.1.1. If the request IP is not in a known range, $\varnothing \leftarrow$ authn(ip_0), then:
                - 3.2.1.1.1. Request the user agent’s public/private credentials, $\{un_0, pw_0\}$.
                - 3.2.1.1.2. If the credentials are not valid, $\varnothing \leftarrow$ authn(un_0, pw_0), then:
                    - 3.2.1.1.2.1. Return an appropriately-formatted error response.
                    - 3.2.1.1.2.2. Terminate the processing of the request.
                - 3.2.1.1.3. If the credentials are valid, $id_0 \leftarrow$ authn(un_0, pw_0), then:
                    - 3.2.1.1.3.1. Establish a new session state, $st_0 = \{ve_0, in_0, id_0, \text{‘challenge’}, now, now + \delta\}$
            - 3.2.1.2. If the request IP address is in a known range, $id_0 \leftarrow$ authn(ip_0), then:
                - 3.2.1.2.1. Establish a new session token, $tk_0 = \{ve_0, in_0, id_0, \text{‘ip’}, now, now + \delta\}$
        - 3.2.2. If an unexpired session state does exist, $\exists\ st_0 = \{\ldots, id_0, \ldots, ex_0\} \mid ex_0 < now$, then:
            - 3.2.2.1. Retrieve the roleset associated with the state identity, id_0: $\{ro_i, \ldots\} \leftarrow$ profile(id_0, ‘role’, $ap_0 \cup$ ‘mrt:any’, $co_0 \cup$ ‘mrt:any’).
            - 3.2.2.2. If at least one rule defined with one of the identity’s roles and a permissive decision does not exist, $\neg\exists\ ru_i \in RS_0 \mid ru_i = \{ro_i, op_0, co_0, ap_0, \text{true}\}$, then:
                - 3.2.2.2.1. Return an appropriately-formatted error response.

3.2.2.2.2. Terminate the processing of the request

3.2.2.3. If at least one rule defined with one of the identity's role and a permissive decision does exist, $\exists\ ru_i \in RS_0 \mid ru_i = \{ro_i, op_0, co_0, ap_0, \text{true}\}$, then:

3.2.2.3.1. Perform the operation.

3.2.2.3.2. Return an appropriately-formatted response reflecting the post-operation status.

3.2.2.3.3. Complete the processing of the request.

4. If a rule with a wildcard role and permissive decision exists, $\exists\ ru_i \in RS_0 \mid ru_i = \{\text{'mrt:any'}, op_0, co_0, ap_0, \text{true}\}$, if which case unauthenticated and unauthorized access is allowed, then:

4.1. Perform the operation.

4.2. Return an appropriately formatted response reflecting the post-operation status.

4.3. Complete the processing of the request.

4 Examples

4.1 Merritt

The Merritt example is based on the following profiles and authorization rules. The rules assume four Merritt operations: read an object, write (i.e., add or update) an object), delete an object, and add user accounts.

Profile

Identity	***Type***	***Application***	***Context***	***Value***
Athos	role	mrt:any	mrt:any	mrt:admin
Porthos	role	Merritt	mrt:any	mrt:admin
Aramis	role	Merritt	UCSF ETD	curator
Aramis	role	Merritt	UCSF image	contributor
D'Artagnan	role	Merritt	UCSF ETD	contributor
D'Artagnan	role	Merritt	UCSF image	curator
Richelieu	role	Merritt	UCSF sound	curator
Planchet				

Authz

Role	***Operation***	***Context***	***Application***	***Decision***
mrt:admin	mrt.any	mrt:any	mrt.any	true
curator	read	mrt:any	Merritt	true
curator	write	mrt:any	Merritt	true
curator	add user	mrt:any	Merritt	true
contributor	write	mrt:any	Merritt	true

mrt:any	read	UCSF ETD	Merritt	true
mrt:any	read	UCSF image	Merritt	true

Note that contexts are defined for specific Merritt collections.

Thus, the complete decision matrix is:

Identity	***UCSF ETD***				***UCSF images***				***UCSF sound***			
	Rea	***Wri***	***Del***	***Add***	***Rea***	***Wri***	***Del***	***Add***	***Rea***	***Wri***	***Del***	***Add***
Athos	true	true	true	true	true	true	true	true	true	true	true	true
Porthos	true	true	true	true	true	true	true	true	true	true	true	true
Aramis	true	true	*false*	true	true	true	*false*	*false*	*false*	*false*	*false*	*false*
D’Artagnan	true	true	*false*	*false*	true	true	*false*	true	*false*	*false*	*false*	*false*
Richelieu	true	false	*false*	*false*	true	false	*false*	false	true	true	*false*	true
Planchet	true	*false*	*false*	*false*	true	*false*	*false*	*false*	*false*	*false*	*false*	*false*
Rochefort	true	*false*	*false*	*false*	true	*false*	*false*	*false*	*false*	*false*	*false*	*false*

In detail:

- Anyone, including Planchet, who is a registered GhOST user but is not associated with *any* Merritt role, and Rochefort, who is not a registered GhOST user and thus represents *anonymous* use of Merritt, can read any object in the UCSF ETD and image collections, without authenticating, by virtue of the rules with wildcard ‘mrt:any’ role for the ‘read’ operation against the two collection contexts.
- Athos can also perform any operation on any object in any collection in Merritt by virtue of holding the ‘mrt:admin’ role for all applications.
- Porthos can also perform any operation on any object in any collection in Merritt by virtue of holding the ‘mrt:admin’ role for the Merritt application.
- Aramis can also write any object and add users to the ETD collection by virtue of being its curator; and write any object in the image collection by virtue of being its contributor.
- D’Artagnan can also write any object and add users to the image collection by virtue of being its curator; and write any object in the ETD collection by virtue of being its contributor.
- Richelieu can also write any object and add users to the sound collection by virtue of being its curator.

Note that the first five rules in the authorization table all reference the wildcard context ‘mrt:any’. This pattern can be used to reduce the number of rules that would otherwise have to be defined for a specific role. Although user agents would generally only be granted the curator role for specific collections, this is enforced by the profile mappings rather than by the authorization rules. Thus, a single rule (with context ‘mrt:any’) can effectively serve the purpose of three rules (with context ‘UCSF ETD’, ‘UCSF images’, and ‘UCSF sound’, respectively). This pattern cannot, however, be used when the role itself is a wildcard, as is the case of the last two rules, granting universal (and unauthenticated) read access to the image and sound collections.

In actuality, Merritt will make use of a more granular set of operations, possibly including:

- Add object
- Update object content
- Update object metadata
- Download object
- Download version
- Download file
- Delete object
- Delete version
- Delete file

5 Implementation

[*More details later…*]

As an optimization, applications making use of GhOST may want to cache rulesets and rolesets locally in order to eliminate the need for subsequent invocations of GhOST functions.

References

[Abbott] Daisy Abbott, *What is Digital Curation?* April 3, 2008 <http://www.dcc.ac.uk/resource/briefing-papers/what-is-digital-curation/>.

[Arms] William Yeo Arms, “Implementing policies for access control,” *D-Lib Magazine* 4:2 (February 1998) <http://www.dlib.org/dlib/februrary98/02arms.html>.

[Denning] Peter J. Denning, Chris Gunderson, and Rich Hayes-Roth, “Evolutionary system development,” *Communications of the ACM* 51:17 (December 2008): 29-31.

[Fisher] David A. Fisher, *An Emergent Perspective on Interoperation in Systems of Systems*, CMU/SEI-2006-TR-003, ESC-TR-2006-003, March 2006 <http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr003.pdf>.

[Foundations] UC3, *Digital Preservation Program: Foundations*, 2010.

[INCITS359] ANSI/INCITS 359-2004, *Information technology – Role based access control*, February 3, 2004 <http://www.techstreet.com/standards/INCITS/359_2004?product_id=1151353>.

[InCommon] InCommon LLC, *InCommon* <http://www.incommonfederation.org/>.

[OpenID] OpenID Foundation, *OpenID Foundation website* <http://openid.net/>.

[Merritt] UC3, *Merritt: An Emergent Approach to Digital Curation Infrastructure*, 2010.

[RFC2119] S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997 <http://www.ietf.org/rfc/rfc2119.txt>.

[Sandhu] Sandhu, Ravi, David Ferraiolo, and Richard Kuhn, "The NIST model for role-based access control: Towards a unified standard," *RBAC '00: Proceedings of the fifth ACM Workshop on Role-Based Access Control*, Berlin, July 26-27, 2000 (New York: ACM, 2000): 47-63 <http://csrc.nist.gov/rbac/sandhu-ferraiolo-kuhn-00.pdf>.

[Shibboleth] Internet2, *Shibboleth* <http://shibboleth.internet2.edu/>.

[UCTrust] University of California, *UCTrust: The University of California Identity Management Federation*, October 30, 2009 <http://www.ucop.edu/irc/itlc/uctrust>.

[XACML] OASIS, *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS standard, February 1, 2005 < http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf >.