P2PSIP Working Group	A. Knauf
Internet-Draft	G. Hege
Intended status: Standards Track	T C. Schmidt
Expires: September 08, 2011	HAW Hamburg
	M. Waehlisch
	link-lab & FU Berlin
	March 07, 2011

A Usage for Shared Resources in RELOAD (ShaRe)
draft-knauf-p2psip-share-00

Abstract

This document defines a RELOAD Usage for shared write access to RELOAD Resources. Shared Resources in RELOAD (ShaRe) form a basic primitive for enabling various coordination and notification schemes among distributed peers. Access in ShaRe is controlled by a hierarchical trust delegation scheme maintained within an access list. A new USER-CHAIN-ACL access policy allows authorized peers to write a Shared Resource without owning its corresponding certificate. This specification also adds mechanisms to store Resources with a variable name which is useful whenever peer-independent rendezvous processes are required.

Status of this Memo

Copyright Notice

described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

This document defines a RELOAD Usage for shared write access to RELOAD Resources and a mechanism to store Resources with a variable name. The Usage for Shared Resources in RELOAD (ShaRe) enables overlay users to share their exclusive write access of specific Resource/Kind pairs with others. Shared Resources form a basic primitive for enabling various coordination and notification schemes among distributed peers. Write permission is controlled by an Access List Kind that maintains a chain of Authorized Peers for a particular Shared Resource. Additionally, this document defines the USER-CHAIN-ACL access control policy that enables a shared write access in RELOAD.

The Usage for Shared Resources in RELOAD is designed for jointly coordinated group applications among distributed peers (c.f. [I-D.knauf-p2psip-disco]). Of particular interest are rendezvous processes, where a single identifier is linked to multiple, dynamic instances of a distributed cooperative service. Shared write access is based on a trust delegation mechanism. It transfers the authorization to write a specific Kind data by storing logical Access Lists. An Access list contains the Kind-ID of the Kind to be shared and contains trust delegations from one authorized to another (previously unauthorized) user.

Shared write access extends the RELOAD security model, which is based on the restriction that peers are only allowed to write resources at a small set of well defined locations (Resource IDs) in the overlay. Using the standard access control rules in RELOAD, these locations are bound to the user name or Node Id in the peer's certificate. This document extends these policies and allows a controlled write access for multiple users at a common Resource Id.

Additionally, this specification defines a new access control policy that enables RELOAD users to store Resources with a variable Resource Name. The USER-PATTERN-MATCH policy allows the storage of Resources whose name complies to a specific pattern. Definition of the pattern is arbitrary, but must contain the user name of the Resource creator.

2. Terminology

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

This document uses the terminology and definitions from the RELOAD base [I-D.ietf-p2psip-base]and the peer-to-peer SIP concepts draft [I-D.ietf-p2psip-concepts]. Additionally, the following terms are used:

Shared Resource: The term Shared Resource in this document defines a RELOAD Resource with its associated Kinds, that can be written or overwritten by multiple RELOAD users following the specifications in this document.

Access List: The term Access List in this document defines a logical list of RELOAD users allowed to write a specific RELOAD Resource/ Kind pair by following the specifications in this document. The list items are stored as Access List Kinds that map trust delegations from user A to user B, where A is allowed to write a Shared Resource and Access List, while B is a user that obtains write access to specified Kinds from A.

Resource Owner: The term Resource Owner in this document defines a RELOAD peer that initially stored a Resource to be shared. The Resource Owner possesses the RELOAD certificate that grants write access to a specific Resource/Kind pair using the RELOAD certificate based access control policies.

Authorized Peer: The term Authorized Peer in this document defines a RELOAD peer that was granted write access to a Shared Resource by permission of the Resource Owner or another Authorized Peer.

3. Shared Resources in RELOAD

A RELOAD user that owns a certificate for writing at a specific overlay location can provide one or more RELOAD Kinds that are designated for a shared write access with other RELOAD users. The mechanism to share those Resource/Kind pairs with a group of users consists of two basic steps. Storage of the Resource/Kind pair to be shared and storage of an Access List to those Kinds. Access Lists are initiated by the Resource Owner and contain Access List items, each delegating the permission to write the shared Kind to a specific user called Authorized Peer. This trust delegation to the Authorized Peer can include the right to further delegate the write permission to the Shared Resource. For each shared Kind data, the Resource owner stores a root item that initiates an Access List. The result is a tree of trust delegations with the Resource Owner as trust anchor.
The Resource/Kind pair to be shared can be any RELOAD Kind that complies to the following specifications:

Separated Data Storage: The specifications in this document ensure that concurrent writing does not effect race conditions. Each data stored within a Shared Resource MUST be exclusively maintained by the RELOAD user that created it. Hence, Usages that allow the

storage of Shared Resources MUST use a RELOAD data model consisting of multiple objects (e.g. Array or Dictionary), each assigned to a single user.

Access Control Policy: To ensure write access to Shared Resource by Authorized Peers, each Usage MUST permit the USER-CHAIN-ACL access policy (see Section 5.4) in addition to its regular access policies (USER-MATCH, USER-NODE-MATCH, etc.).

user_name field: To identify the originator of a stored value, the Kind data structure of a Resource allowing shared write access MUST define a <0..2^16-1> long opaque user_name value. It contains the user name value of the RELOAD certificate which was used to store and sign Kind data. The user_name field allows any consumer of the data to request the public key certificate of the originator of the stored data and to verify its provenance and integrity.

4. Access List Definition

4.1. Access List

An Access List in this document specifies a logical list of AccessList data structures defined in Section 4.2. Each entry delegates write access to specific Kind data and is stored at the same overlay location as the Shared Resource. It allows the RELOAD user who is authorized to write at a specific Resources-ID to delegate his exclusive write access for the specified Kinds to further users of a RELOAD instance. Each Access List data structure therefore carries the information about who delegates write access to whom, the Kind-ID of the Resource to be shared, and whether delegation includes write access to the Access List itself. The latter condition grants the right to delegate write access further for an Authorized Peer. Access Lists are stored within a RELOAD array data model and are initially created by the Resource Owner.
Figure 1 shows an example of an Access List. The array entry at index #0 displays the initial storage of an Access list to a Shared Resource with Kind-ID 1234 at the same Resource-ID. It represents the root item of the trust delegation tree to the shared RELOAD Kind and initiates an Access List to the specified Kind data. The root entry MUST contain the mapping from Resource owner to Resource owner and MUST only be written by the owner of the public key certificate to this Resource-ID.
The array entry at index #1 represents the first trust delegation to an Authorized peer that is permitted write access to the Shared Resource with Kind-ID 1234. Additionally, the Authorized peer Alice is also granted write access to the Access List as indicated by the allow_delegation flag (AD) set to 1. It authorizes Alice to store further trust delegations to the Shared Resource, respectively, store items into the Access List. For instance, Alice permits Bob to access the Shared Resource, but Bob in turn is not allowed to write the Access

List (AD = 0). The Authorized Peer Alice signs the Access List item with her own private key.
In order to share multiple Kinds at a single location, the Resource Owner can initiate new Access Lists that are referencing to another Kind-IDs as shown in array entry index #42. Note that overwriting existing items in an Access List that reference a different Kind-ID, revokes all succeeding trust delegations in the tree. Hence, Authorized Peers are not enabled to overwrite any existing Access List item (see Section 5.2). The Resource Owner is allowed to overwrite existing Access List items, but should be aware of its consequences.

Access List			
#	Array Entries	AD	Signature
0	Kind:1234 from:Owner -> to:Owner	1	signed by Owner
1	Kind:1234 from:Owner -> to:Alice	1	signed by Owner
2	Kind:1234 from:Alice -> to:Bob	0	signed by Alice
...	...	...	...
42	Kind:4321 from:Owner -> to:Owner	1	signed by Owner
43	Kind:4321 from:Owner -> to:Carol	0	signed by Owner

Implementations of ShaRe should be aware that the trust delegation in an Access List is not loop free per se. Self-contained circular trust delegation from A to B and B to A are possible, even though not very meaningful.

4.2. Data Structure

The Kind data structure for the access list is defined as follows:

```
struct {
  opaque resource_name<0..2^16-1>;
  KindId kind;
  opaque from_user<0..2^16-1>;
  opaque to_user<0..2^16-1>;
  Boolean allow_delegation;
} AccessListData;

struct {
  uint16 length;
  AccessListData data;
} AccessListItem;
```

The AccessListItem structure is composed of:

length:
    Length of the Access List data structure
data:
    Data of the Access List

The content of the AccessListData structure is defined as follows:

resource_name: This opaque string contains the Resource Name of the Shared Resource in an opaque string. Thus, the AccessListData meet the requirements for the USER-PATTERN-MATCH access policy (see Section 6.1).

kind: This field contains the Kind-ID of the Kind that will be shared.

from_user: This field contains the user name of that RELOAD peer the grants write permission to the Shared Resource. The user name is stored as an opaque string and contains the user name value of the certificate that is associated with the private key that signed this Access List item.

to_user: This field contains the user name of the RELOAD peer that obtains writing permission to the Shared Resource.

allow_delegation: This Boolean flag indicates if true, that the Authorized peer in the 'to_user' field is allowed write access to the Access List in order to delegate the write permission to the Shared Resource to further users.

The ACCESS-LIST kind is defined as follows:

Name ACCESS-LIST

Data model The Data model for the ACCESS-LIST data is array.

Access Control

Initial storages of ACCESS-LIST data by the Resource Owner use the same Access Control Policy as the Shared Resource. For instance, if the access policy for the Shared Resource is USER-NODE-MATCH, then the access policy for the ACCESS-LIST data is USER-NODE-MATCH. Storages by Authorized Peers use the USER-CHAIN-ACL access policy (see Section 5.4).

5. Access Control to Shared Resources

5.1. Granting Write Access

Write access to a Kind that is intended to be shared with other RELOAD users can solely be issued by the Resource Owner. If the Resource owner shares an existing Resource/Kind pair, it should ensure that it does not unintentionally overwrite an existing Access List item. Hence, before sharing the Resource, its owner performs a fetch request for the Access List Kind that requests the entire array. If the retrieved array does not contain an Access List root item to the desired Kind, the Resource Owner stores a new root item for the desired Kind-ID and sets the AccessListData vales 'from_user' and 'to_user' to the user name of the Resource Owner. If an Access List root item exists, the Resource Owner delegates write access by storing an Access List item setting the 'from_user' to its user name and setting the 'to_user' equal to the name of the RELOAD user that obtains write access.
If an Authorized Peer intents to delegate write access to a Shared Resource, it likewise fetches the entire array of the Access List Kind to prevent an unauthorized write attempt to an existing Access List item. Afterwards it delegates write access to the specified Kind by storing an Access List item setting the 'from_user' value to its own user name and setting the 'to_user' value to RELOAD user that obtains write access. Note, that an Authorized Peer is only allowed to add items into an Access List it is registered in with the 'allow_delegation' flag set to true.

5.2. Revoking Write Access

Write permissions MAY be revoked by storing a non-existent value [I-D.ietf-p2psip-base] to the corresponding item in the Access Control List. A revoked permission automatically invalidates all delegations performed by that user and also all subsequent delegations. This allows to invalidate entire subtrees of the delegations tree with only a single operation. Overwriting the root item with a non-existent value of an Access List invalidates the entire delegations tree.
An Access List item MUST only be written by the user who initially stored the corresponding entry. The only exception is by the Resource Owner that is allowed to overwrite Access list items at all times with a non-existent value for revoking write access.

5.3. Storage and Validation

5.3.1. Operations of the Storing Peer

The storing peer (the peer at which Shared Resource and Access List are physically stored) is responsible for enforcing the correct access policy when it is requested to store values of a Shared Resource. The storing peer first checks, whether the request is signed with the private key that corresponds to a certificate valid for this Resource-ID as enforced by the standard access policies (USER-MATCH, USER-NODE-MATCH, etc.) defined in the RELOAD base protocol [I-D.ietf-p2psip-base], or the policy USER-PATTERN-MATCH defined in this document (see Section 6.1).
If not, the storing peer continues by checking whether any of the received RELOAD Kinds of the store request allows the USER-CHAIN-ACL access control policy. If so, the storing peer fetches the Access Lists for those Kinds and enforce the USER-CHAIN-ACL access policy (see Section 5.4). Since the Access list MUST be stored at the same overlay location as the Shared Resource, this operation is a local lookup. Analogously, a storing peer that is requested to store an Access List Kind first verifies whether the requester is allowed to store values at this Resource-ID by its certificate. Otherwise the storing peer MUST locally fetch the Access List of the requested Resource Id and enforce the USER-CHAIN-ACL policy.

5.3.2. Operations of the Accessing Peer

An accessing peer (a RELOAD peer that fetches a Shared Resource) SHOULD validate the provenance and integrity of a retrieved data value and the authorization of the data originator. The latter is verified using the Access List Kind. The accessing peer requests all Access Lists that are stored under the same Resource-ID as the Shared Resource by requesting the entire array range. This request could be sent in the same fetch request as the request for the Shared Resource. The accessing peer then checks, whether any of these Access Lists refers to the Kind of Shared Resource by its Kind-ID. If true, the accessing peer compares the 'to_user' value of each Access List item with the mandatory user_name value of the Shared Resource for equality. If the comparison fails, the accessing peer MUST ignore the data of the retrieved Shared Resource. Else, the accessing peer repeats this comparison with the value of the 'from_user' field of this item with each 'to_user' field of the Access List. This procedure continues until both 'from_user' and 'to_user' values are equal. The accessing peer then hashes the 'from_user' using the hash function of the overlay algorithm. If the hash is equal to the Resource-ID of the Shared Resource, the authority of the originator of the stored data is validated. The accessing peer then proceeds with the provenance and integrity tests.
The accessing peer verifies provenance and integrity of the retrieved kind data using the certificate corresponding to the mandatory

user_name field of the Shared Resource entry. The certificate can be retrieved by applying the Certificate Usage [I-D.ietf-p2psip-base] or other means (e.g., caching from a previous request).
The accessing peer MAY cache previously fetched Access List to a maximum of the individual items' lifetimes. Since stored values could have been changed or invalidated prior to their expiration an accessing peer uses a stat request to check for updates before using the cached data. If a change has been detected it fetches the latest Access List.

5.4. USER-CHAIN-ACL Access Policy

This document specifies an additional access control policy to the RELOAD base draft [I-D.ietf-p2psip-base]. The USER-CHAIN-ACL policy allows Authorized Peers to write a Shared Resource, even though they do not own the corresponding certificate. Access is controlled by the values stored within the Access List Kind that explicitly permits Authorized Peers writing access to Shared Resources or the Access List (or both) by their user name. Hence, if a request in not signed with a private key that allows write access to a Resource by any access control policy defined in the RELOAD base specification, a storing peer MUST enforce the USER-CHAIN-ACL policy:
When accessing the Shared Resource, a given value MUST be written or overwritten if and only if the request is signed with a key that is associated with a certificate whose user name is stored in any 'to_user' value of an Access List associated to the Shared Resource. If true, this comparison has to be repeated for the 'from_user' value of that Access List item with each other 'to_user' value in this Access List. This procedure continuous until 'from_user' and to 'to_user' are equal. Then, if the hash over the 'from_user' equals the Resource-ID, the requester is authorized to write the Shared Resource.
When accessing the Access List, a given value MUST be written or overwritten if and only if the request is signed with a key that is associated with a certificate whose user name is stored in any 'to_user' value in the same Access List as the requested Access List. Additionally, the 'allow_delegate' value of this Access List item MUST be set true. If this query is successes, the comparison has to be repeated for the 'from_user' value of that Access List item with each other 'to_user' value in the Access List. This procedure continuous until 'from_user' and to 'to_user' are equal. Then, if the hash over the 'from_user' equals the Resource-ID, the requester is authorized to write the Access List.

6. Extension for Variable Resource Names

In certain use cases such as conferencing (c.f. [I-D.knauf-p2psip-disco]) it is desirable to extend the set of Resource Names and thus Resource-IDs a peer is allowed to write beyond those defined through the user name or NodeId fields in its certificate. This is accomplished by the USER-PATTERN-MATCH access policy described here.

Each RELOAD node uses a certificate to identify itself using its user name (or Node-ID) while storing data under a specific Resource-ID. The USER-PATTERN-MATCH scheme follows this paradigm by allowing to store values whose Resource Name is derived from the user name in the certificate of a RELOAD peer, but extends the set of allowed Resource Names. This is done by using a Resource Name which contains a variable substring but matches the user name in the certificate using a pattern defined in the configuration document. Thus despite being variable an allowed Resource Name is closely related to the Owner's certificate. A sample pattern might be formed as the following:

Example Pattern:
.*-conf-$USER@$DOMAIN

When defining the pattern care must be taken that no conflict arises for two user names of witch one is a substring of the other. In this case the peer with the name which is the substring could choose the variable part of the Resource Name so that the resulting string contains the whole other user name and thus he could write the other user's resources. This can easily be prevented by delimiting the variable part of the pattern from the user name part by some fixed string, that is usually not part of a user name (e.g. the "-conf-" in the above Example).

6.1. USER-PATTERN-MATCH Access Policy

Thus, using the USER-PATTERN-MATCH policy, a given value MUST be written or overwritten if and only if the request is signed with a key that is associated with a certificate whose user name matches the Resource Name using the pattern specified in the configuration document. The Resource Name MUST be taken from an opaque resource_name field in the corresponding Kind data structure. Hence, each RELOAD Usage that utilizes the USER-PATTERN-MATCH policy, MUST define an opaque resource_name field within the Kind data structure, that contains the Resource Name whose hash equals the Resource-ID.

6.2. Overlay Configuration Document Extension

This document extends the overlay configuration document by defining new elements for patterns relating resource names to user names.
The <variable-resource-names> element serves as a container for one or multiple <pattern> sub-elements.
Each <pattern> element defines the pattern to be used for a single Kind. It is of type xsd:string, which is interpreted as a regular expression. In the regular expression $USER and $DOMAIN are used as variables for the corresponding parts of the string in the certificate user name field ($USER before and $DOMAIN after the '@'). Both variables MUST be present in any given pattern. The <pattern> element

has the attribute "kind" which contains the Kind name for which this pattern is used.
A <pattern> element MUST be present for every Kind for which the variable resource names extension is allowed in an overlay.
The Relax NG Grammar for the Variable Resource Names Extension is:

```
<!--
    VARIABLE RESOURCE NAMES ELEMENT
-->
parameter &= element variable-resource-names {
    <!--
        RESOURCE NAME PATTERN ELEMENT
    -->
    element pattern {
        attribute kind { xsd:string },
        xsd:string
    }*
}?
```

7. Security Considerations

In this section we discuss security issues that are relevant to the usage of shared resources in RELOAD.

7.1. Resource Exhaustion

Joining a RELOAD overlay inherently poses a certain resource load on a peer, because it has to store and forward data for other peers. In common RELOAD semantics, each ResourceId and thus position in the overlay may only be written by a limited set of peers - often even only a single peer, which limits this burden. In the case of Shared Resources, a single resource may be written by multiple peers, who may even write an arbitrary number of entries (e.g., delegations in the ACL). This leads to an enhanced use of resources at individual overlay nodes. The problem of resource exhaustion can easily be mitigated for Usages based on the ShaRe-Usage by imposing restrictions on the maximum number of entries a single peer is allowed to write at a single location.

7.2. Malicious or Misbehaving Storing Peer

The RELOAD overlay is designed to operate despite the presence of a small set of misbehaving peers. This is not different for Shared Resources since a small set of malicious peers does not disrupt the functionality of the overlay in general, but may have implications for the peers needing to store or access information at the specific locations in the ID space controlled by a malicious peer. A storing peer could withhold stored data which results in a denial of service to the group using the specific resource. But it could not return forged

data, since the validity of any stored data can be independently verified using the attached signatures.

7.3. Privacy Issues

All data stored in the Shared Resource is publicly readable, thus applications requiring privacy need to encrypt the data. The ACL needs to be stored unencrypted, thus the list members of a group using a Shared Resource will always be publicly visible.

8. IANA Considerations

TODO: register Kind-ID code point at the IANA

9. Acknowledgments

This work was stimulated by fruitful discussions in the P2PSIP working group and SAM research group. We would like to thank all active members for constructive thoughts and feedback. This work was party funded by the German Federal Ministry of Education and Research, projects HAMcast and Mindstone.

10. References

10.1. Normative References

[I-D.ietf-p2psip-base]	Jennings, C, Lowekamp, B, Rescorla, E, Baset, S and H Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", Internet-Draft draft-ietf-p2psip-base-19, October 2011.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[I-D.ietf-p2psip-concepts]	Bryan, D, Matthews, P, Shim, E, Willis, D and S Dawkins, "Concepts and Terminology for Peer to Peer SIP", Internet-Draft draft-ietf-p2psip-concepts-04, October 2011.
[I-D.knauf-p2psip-disco]	Knauf, A, Hege, G, Schmidt, T and M Waehlisch, "A RELOAD Usage for Distributed Conference Control (DisCo)", Internet-Draft draft-knauf-p2psip-disco-04, October 2011.

Authors' Addresses

Alexander Knauf Knauf HAW Hamburg Berliner Tor 7 Hamburg, D-20099 Germany Phone: +4940428758067 EMail: alexander.knauf@haw-hamburg.de URI: http://inet.cpt.haw-hamburg.de/members/knauf

Gabriel Hege Hege HAW Hamburg Berliner Tor 7 Hamburg, D-20099 Germany Phone: +4940428758067 EMail: hege@fhtw-berlin.de URI: http://inet.cpt.haw-hamburg.de/members/hege

Thomas C. Schmidt Schmidt HAW Hamburg Berliner Tor 7 Hamburg, D-20099 Germany EMail: schmidt@informatik.haw-hamburg.de URI: http://inet.cpt.haw-hamburg.de/members/schmidt

Matthias Waehlisch Waehlisch link-lab & FU Berlin Hoenower Str. 35 Berlin, D-10318 Germany EMail: mw@link-lab.net URI: http://www.inf.fu-berlin.de/~waehl