High level design of Capability in Colibri

By Huang Hua <hua_huang@xyratex.com>
Date: 2010/11/1
Revision: 0.1

[Text in square brackets and with a light-green background is a commentary explaining the structure of a design document. The rest is a fictional design document, used as a running example.]

This document presents a high level design (HLD) of an authentication based on capability in Colibri C2 core. The main purposes of this document are: (i) to be inspected by C2 architects and peer designers to ascertain that high level design is aligned with C2 architecture and other designs, and contains no defects, (ii) to be a source of material for Active Reviews of Intermediate Design (ARID) and detailed level design (DLD) of the same component, (iii) to serve as a design reference document.

The intended audience of this document consists of C2 customers, architects, designers and developers.

0. Introduction

In a distributed system, authentication is essential. Authentication between different components, locally or remotely, are used as permission checking, while the system handles operations.

Authentication, based on capability[0], will be used in Colibri.

Right now in this document, we will present a stub for capability-based authentication for Colibri.

1. Definitions

- Colibri Object. This may be global object (file), or component object (file), or locks, layouts, etc.
- Capability. This is a permission assigned to an object, which allows some specific operations to be carried on this object from some specific user.

2. Requirements

- [r.capa.issuer] capability is issued by some server (meta-data server, or data server). The server is considered to be trusted entity in the system.
- [r.capa.owner] capability is tied to some object and user.
- [r.capa.expire] capability will expire at sometime.
- [r.capa.renew] capability can be renewed from client.
- [r.capa.authenticate] capability can be authenticated by server, maybe different from the issuer of this capability. This is called remote authentication.
- [r.capa.communicate] capability along with object can be transferred from one node to another.
- [r.capa.encypted] capability transferred to client is encypted. It is opaque to client. Client cannot decrypt to reveal the object owner and user.
- [r.capa.type] various capability flavors are supported: null, omg (owner-group-mode), compound.

3. Design highlights

There is a capability master on a node. Capabilities for objects(or component objects) are issued by this master, and can be authenticated later.

4. Functional specification

Capabilities are computed and hashed by master for object, access permissions, and users. Capabilities can also be authenticated by master later. Capabilities can be renewed here.

5. Logical specification

Capability is an array of bytes. This is opaque to any other components in the system. Capability is calculated from object identifier, access permission, user identifier and time-stamp, and then hashed by the master.
Currently as to define a stub for the system, this will be left empty.
More detailed capability design will be done later.

5.1. Conformance

[For every requirement in the Requirements section, this sub-section explicitly describes how the requirement is discharged by the design. This section is part of a requirements tracking mechanism, so it should be formatted in some way suitable for (semi-)automatic processing.]

5.2. Dependencies

N/A

5.3. Security model

This capability is for security.

5.4. Refinement

N/A

6. State

N/A

6.1. States, events, transitions

N/A

6.2. State invariants

N/A

6.3. Concurrency control

N/A

7. Use cases

[This section describes how the component interacts with rest of the system and with the outside world.]

7.1. Scenarios

[This sub-section enumerates important use cases (to be later used as seed scenarios for ARID) and describes them in terms of logical specification.]

Scenario	usecase.capa.issue
Relevant quality attributes	usability
Stimulus	an incoming request
Stimulus source	client operation
Environment	server
Artifact	a capability.
Response	a capability is issued.
Response measure	correct capability is issued.
Questions and issues	

Scenario	usecase.capa.renew
Relevant quality attributes	usability
Stimulus	timeout of some capability
Stimulus source	client
Environment	server
Artifact	a new capability.
Response	a capability is renewed.
Response measure	correct capability is renewed.
Questions and issues	

Scenario	usecase.capa.authenticate
Relevant quality attributes	usability
Stimulus	an incoming request
Stimulus source	client operation
Environment	server
Artifact	permission
Response	permission is granted or rejected based on the capability and object attr.
Response measure	correct permission is granted.
Questions and issues	

[UML use case diagram can be used to describe a use case.]

7.2. Failures

N/A

8. Analysis

8.1. Scalability

N/A

8.2. Other

N/A

8.2. Rationale

N/A

9. Deployment

9.1. Compatibility

N/A

9.1.1. Network

9.1.2. Persistent storage

9.1.3. Core

N/A

9.2. Installation

N/A

10. References

[0] http://en.wikipedia.org/wiki/Capability-based_security

11. Inspection process data

[The tables below are filled in by design inspectors and reviewers. Measurements and defects are transferred to the appropriate project data-bases as necessary.]

11.1. Logt

	Task	Phase	Part	Date	Planned time (min.)	Actual time (min.)	Comments
umka	capa-sub	HLDINSP	1		180		
Nathan	capa-sub	HLDINSP	1		180		

11.2. Logd

No.	Task	Summary	Reported by	Date reported	Comments
1	capa-sub				
2	capa-sub				
3	capa-sub				
4	capa-sub				
5	capa-sub				
6	capa-sub				
7	capa-sub				
8	capa-sub				
9	capa-sub				