Attribute-Based Access Control (ABAC) for Hyperledger Fabric

Author: Keith Smith

# Overview

The purpose of this document is to describe how to add support for ABAC (Attribute-Based Access Control) and RBAC (Role-Based Access Control) to Hyperledger Fabric.

# Goals

1. Support ABAC for chaincode.
2. Use ABAC for fabric code. In particular, use it to provide another option which does not require populating the msp/admincerts directory.
3. Discuss RBAC.

# 

# Implementing ABAC for chaincode

This section describes the changes to 3 components in order to implement ABAC for chaincode: the fabric-ca-server, the chaincode API, and optionally to the SDKs.

### Changes to fabric-ca

The fabric-ca-server already allows associating arbitrary user-defined attributes with an identity via registration. It also supports issuing enrollment certificates (ECerts). In order to support ABAC, enhance the fabric-ca-server to optionally insert one or more attributes into an ECert via an x509 extension.

**For v1.1**, fabric-ca-server will not support attribute encryption. This means both the name and value of the attribute will always be “in the clear” inside the ECert. **Post v1.1**, fabric-ca-server would support attribute encryption using a TBD mechanism for key distribution.

The list of default attributes to add to an ECert may be specified at registration time. This default list may be overridden at enrollment time.

The **fabric-ca-client register** command is enhanced to allow an attribute to be marked as one of the default attributes to add to an ECert. For example, the optional “:ecert” at the end of the following command marks attribute ‘attr1’ to be added to user1’s ECert if no specific attributes are requested at enrollment time.

# fabric-ca-client register --id.name user1 --id.secret user1pw --id.type user --id.affiliation org1 -id.attrs attr1=val1:ecert

The fabric-ca-server is also enhanced to restrict which attributes may be registered by a registrar. In particular, a registrar may register an attribute if one of the following is true:

1. The registrar owns the attribute of the same name

2. The registrar owns the **hf.Registrar.Attributes** attribute with values that matches the attribute or attribute pattern being registered. For example, if the registrar has the **hf.Registrar.Attributes** attribute with a value of “a.b.\*, c.d”, the registrar may register attributes whose names starts with “a.b.” or which equals “c.d” even though the registrar does not own these attributes. If an identity has the **hf.Registrar.Attributes** attribute with a value of “\*”, the identity can register any attribute.

Another special check is made when registering a new identity with the **hf.Registrar.Attributes** attribute. In particular, the value of the registrar’s hf.Registrar.Attributes attribute must be equal to or a superset of the value of the registree’s hf.Registrar.Attributes. This prevents the registrar from registering an identity with more attribute privileges than it owns itself.

### Library API

The following library APIs will be added to the go SDK.

import "github.com/hyperledger/fabric/core/chaincode/shim"

// NewClient constructs a client representing the chaincode invoker identity

NewClient(stub shim.ChaincodeStubInterface) (Client, error)

// ClientIdentity represents information about the identity that submitted the

// transaction

type ClientIdentity interface {

// GetID returns the ID associated with the invoking identity. This ID

// is guaranteed to be unique within the MSP. It returns “” if the

// identity is anonymous and the ID is not revealed.

GetID() (string, error)

// Return the MSP ID of the client

GetMSPID() (string, error)

// GetAttributeValue returns the value of the client's attribute named

// `attrName`. If the client possesses the attribute, `found` is true and

// `value` equals the value of the attribute. If the client does not possess

// the attribute, `found` is false and `value` equals "".

GetAttributeValue(attrName string) (value string, found bool, err error)

// AssertAttributeValue verifies that the client has the attribute named

// `attrName` with a value of `attrValue`; otherwise, an error is returned.

AssertAttributeValue(attrName, attrValue string) error

// GetX509Certificate returns the X509 certificate associated with the client,

// or nil if it was not identified by an X509 certificate.

GetX509Certificate() (\*x509.Certificate, error)

}

This would allow the chaincode developer to make authorization decisions based on an attribute value and/or the MSPID associated with the client. For example, if only an identity associated with a particular MSP and with a specific attribute is permitted to perform some operation, the MSPID and the attribute value may be checked.

### Other changes to the SDKs

The other changes to the SDKs are very small, but are also optional. If the “fabric-ca-client register” command is used as described earlier to mark an attribute to be added to an ECert by default, then the SDK need not request that attribute in the enrollment request.

That said, the body of the enrollment and reenrollment requests which the SDK sends to the fabric-ca-server has an additional field as follows. This should be exposed via the SDK’s API.

{

…

“attr\_reqs”: [

{“name”: “attr1”, “require”: true},

{“name”: “attr2”, “require”, false}

]

}

In this example, the SDK is requesting that two attributes be added to the ECert: “attr1” and “attr2”. If the value of “require” is true and the identity does not possess the attribute, an error is returned; otherwise, the attribute request is skipped which means the attribute was optional. So if the identity has attribute “attr1” but not “attr2”, the ECert will still be issued without error.

Suppose for example that you have application chaincode which must only be invoked by identities with the **accountID** attribute. Setting “require” to true for this attribute is useful to fail fast, so that the certificate will not be issued; however, you still need to check for the existence of this attribute in chaincode also since a client may have obtained a certificate via another means.

# Using ABAC in fabric code

Populating the msp/admincerts folder is difficult administratively. This could be changed by using ABAC as was described in https://jira.hyperledger.org/browse/FAB-3752.

In particular, we enhance the SatisfiesPrincipal function of msp to do more than a byte comparison. Logically speaking, change the comparison from "if the caller's cert exactly matches any of these admincerts" to "if the caller's cert exactly matches any of these admincerts OR (if the caller's cert is issued by any of these admincerts AND the cert has the 'hf.admin' attribute with a value of 'true').

# RBAC

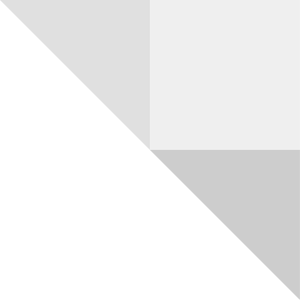
RBAC (Role-Based Access Control) answers the question, “Does the identity have this role?”. When the identity is represented by an X509 certificate, it could be based on anything in the certificate. For example, it could be very simply implemented by checking to see if the identity possesses an attribute with the same name as the role name, and is the value of the attribute “true”. However, this would require that external CAs other than fabric CA issue certificates with the same custom extension, which is not likely.

Really, any standard fields in the certificate could be used. For example, OUs could be used. However, OUs are different in that they are hierarchical and order dependent, much like the path to a file on the file system. For example, consider the following two DNs, where Alice is an administrator in the finance department and Bob is an administrator in the engineering department.

1. C=US, O=Org1, OU=Finance, OU=Admin, CN=Alice
2. C=US, O=Org1, OU=Engineering, OU=Admin, CN=Bob

If we want to specify all administrators in the finance department, it would not be sufficient to include all DNs with OU=Admin; otherwise, Bob would also be included. We would need to specify:

C=US, O=Org1, OU=Finance, OU=Admin

I think a general purpose expression evaluation language is needed. If we try to support other specific syntax languages, then we end up having difficulty extending the language and we will keep adding specific configuration that we have to support forever.

For example, the following are sample expressions with a description of what they mean:

1. Match any certificate with a distinguished name which matches the regular expression, so any Admin in the Finance department of Org1:

**DN like ‘C=US, O=Org1, OU=Finance, OU=Admin, CN=\*’**

1. Anyone from MSP ‘org1’ or ‘org2’ with the “admin” attribute of true

**MSP in [‘Org1’,’Org2’] AND ATTR(‘admin’) = ‘true’**

NOTE: See <https://docs.oracle.com/cd/E24191_01/common/tutorials/authz_cert_attributes.html> for some related discussion about how OUs are used.