AUTHORIZATION FRAMEWORK FOR SERVICE ORIENTED ARCHITECURE

Zsolt KOCSIS
Advisor: András Pataricza

I. Introduction

Setting up secure application architecture is very challenging. The Service Oriented Architecture design allows and requires centrally manageable security services, among them the authorization service is the key to build model based security infrastructure. Although the theory of different security models are well-known, the definition and coding of the authorization rules are not complicated, due to the lack of robustness and high performance the solutions based on these models are still not in everyday use. This paper introduces an industry scale solution based on standard Tivoli technology to set up a universal authorization service.

II. Authorization engine based on IBM Tivoli technology

There are several security models developed like the Bell- La Padula, Clark- Wilson model, or Role Base Access Control[2].Common in these models is that the runtime evaluations of the necessary rules are very resource consuming, considering that each object access needs to be verified.

The solution is based on Tivoli Access Manager (TAM). The TAM uses ACLs to evaluate the access requests, provides an authentication framework, and includes a robust authorization engine. Nevertheless the solution does not support the evaluation a business condition.

The standard TAM provides the following authorization logic:

- Role Based (standard ACLs)
- Protected Object Policy - POP
- Rule Based (dynamic condition evaluation)

The rule evaluation is rather slow, the POP conditions are rather limited, therefore we had to work out a solution being able to evaluate flexible business logic decisions and providing central authorization service to make Allow-Deny decisions.

An application model is composed of the following objects:

- Business objects – these are the protected objects
- Operations - business operations available on the Business Objects
- Users in Roles

Special runtime conditions (business conditions) must be evaluated before each authorization decision. These decisions are necessary to define which users can execute a given operation on an object, and under what conditions

A very important development criteria was that the authorization engine must be powerful enough to serve a very high load of authorization requests. I designed a special, stored result Boolean arithmetic to solve the problem, implemented as special ACLs in the TAM's authorization database, serving the authorization request through the provided authorization API.

The solution is able to evaluate request calls passing the grouped Boolean or Integer runtime conditions, make virtually any arithmetic operation with these conditions, and provide the Allow-Deny result.

We worked out a special management interface - as an add-on to the TAM - to allow defining the special rules. It processes the rules-input as human-readable regular logical expressions and uploads them as special ACLs to TAM.

The application itself contains a small plug-in that converts the runtime conditions received from the calling application into standard TAM requests and receives the formal authorization result. All authorization decisions are made with the native TAM throughput.

There was one TAM feature to consider: it is the 32 bit internal word -length. This is a bottleneck to make the solution fully generic on one hand, but on the other hand with the evaluation method detailed below the solution is very well usable in most environments.

I defined the following building elements to implement the condition evaluation engine:

a. Condition Group

Basic building block, implemented as a special TAM ACL type.
One *condition group* means five binary variable. Within one condition group any combination of the conditions can result in Allow decision.
Input range: 5 bit Boolean, or 0-31 Integer, or any combination.
Sample : [A] = (A and not B) or C or D and not E , or N=1,2,3, N<17 etc.
Output: Boolean, Yes/No
Number of calls: 1 call

b.Condition Chain

To one protected object several (max. 32) *condition groups* can be attached *in chain* ,and the result of the evaluation for all groups can be performed with a single system call.
The result is a logical AND for all the chained condition groups.
Sample: {ABC} = [A] and [B] and [C]
Output: Boolean, Yes / No
Number of calls: 1 call / chain

c. Condition Relation

Relations are freely definable based on the result of five formerly evaluated Condition Chains. This is very flexible, all exeption like condition can be accomplished by this module.
Sample CR= {A} and {B) or ({C} and {D}))and not {E}, etc.
Number of calls: 6 calls

A. Model Based Security

We set up a working version of the environment – including a simple test application sending authorization requests. The framework itself is general enough to provide a solution to implement different security models. We plan that having this framework we can start to implement the model based verification according to a given security model. As the first step we are going to start with the Wilson-Clark authorization model.

B. Authorization service for Service Oriented Architecture

The authorization engine layer is easily accessible from any application requiring authorization decisions , therefore the solution provides a framework to externalize the definition and evaluation of all authorization requests needed. As a further step the definition of this service will be done, and the solution will be tested in live environment.

References

[1] John Ganci, Hinrich Boog ,Melanie Fletcher,Brett Gordon : *Develop and Deploy a Secure Portal Solution*, IBM Redbook 2004

[2] David Clark, David Wilson:*A Comparison of Commercial and Military Computer Security Policies*, Proceedings of the IEEE Computer Society Symposium on Security and Privacy, Oakland, 1987.