一种基于属性的企业云存储访问控制方案*

熊 智, 王 平, 徐江燕, 蔡伟鸿
(汕头大学 计算机科学与技术系, 广东 汕头 515063)

摘 要: 针对企业云存储应用的需求,以及目前基于属性的访问控制方案在访问规则描述方面存在的不足,提出了一种适合企业云存储的基于属性的访问控制方案。该方案直接用字典变量表示主体、资源和环境实体,用 Python 逻辑表达式描述访问规则,并采用 eval 函数执行规则,从而使访问规则容易编写,表达能力强,且执行开销小。考虑到资源的层次结构特点,分别为不同的访问权限设计了相应的访问规则继承策略以简化访问规则的编写,并采用跨语言的服务开发框架 Thrift 对访问控制器进行了实现。

关键词: 访问控制; 基于属性; 访问规则; 企业云存储; 层次结构
中图分类号: TP309 **文献标志码**: A **文章编号**: 1001-3695(2013)02-0513-05
doi:10.3969/j.issn.1001-3695.2013.02.054

Attribute based access control strategy for enterprise cloud storage

XIONG Zhi, WANG Ping, XU Jiang-yan, CAI Wei-hong
(*Dept. of Computer Science & Technology, Shantou University, Shantou Guangdong* 515063, *China*)

Abstract: To satisfy the demand of enterprise cloud storage application, and to overcome the drawbacks of the existing attri-bute based access control strategy in access rule description, this paper proposed an attribute based access control strategy suitable for enterprise cloud storage application. The strategy used dictionary variables to denote subject, resource, and environment entities, adopted Python logical expression to express access rule, and adopted the eval function to execute rule; therefore, access rule was easy to write, had strong expression ability, and costed a little. Considering the hierarchical structure of the resources, the strategy designed corresponding access rule inheritance scheme for different access rights to simplify the writing of access rules. It used a cross-language services development framework, Thrift, to implement the access control server.

Key words: access control; attribute based; access rule; enterprise cloud storage; hierarchical structure

0 引言

企业云存储,又称企业网盘,是基于云计算理念推出的企业数据网络存储和管理解决方案,利用互联网后台数据中心的海量计算和存储能力为企业提供数据汇总分发、存储备份和管理等服务。企业云存储必须保证企业数据的安全性,包括保密性、完整性和可用性。其中数据的保密性又包括两个方面:a)对数据进行加密,确保非本企业的用户(被授权的除外)以及云存储运营商都无法获得本企业数据的明文;b)对数据进行访问控制,确保只有被授权的本企业用户才能访问特定的数据,这也是本文研究的内容。

目前市场上比较流行的企业云存储产品有联想企业网盘、网易企业网盘和 115 企业网盘等。它们的访问控制方案基本上都是采用自主访问控制(discretionary access control,DAC)和基于角色访问控制(role based access control,RBAC)模型。但这两种模型无法满足企业云存储的应用需求,主要表现为:a)随着用户和资源数量的增长,DAC 中访问控制列表(access control list,ACL)的规模急剧增加,难以管理和维护;b)若要进行细粒度的访问控制,必须对用户进行精确区分,从而使 RBAC 需要定义大量的用户角色,这给角色的分配和管理带来困难;c)它们都是静态的授权机制,没有考虑上下文环境对授权的影响。

基于属性的访问控制(attribute based access control,ABAC)是将相关实体(如主体、资源和环境等)的属性作为授权的基础来进行访问决策,它能解决复杂信息系统中的细粒度访问控制和大规模用户动态扩展的问题[1],为开放网络环境提供了较理想的访问控制方案。特别地,如果将身份、角色以及资源安全密级等信息也抽象化为实体属性,ABAC 则能实现传统 IBAC(identity based access control,基于身份的访问控制)、RBAC 和 MAC(mandatory access control,强制访问控制)模型的功能[2]。

ABAC 基于主体、资源和环境等实体的属性来制定访问规则(或策略),并依照访问规则来进行访问决策,因而访问规则的描述是 ABAC 研究的关键问题之一。XACML(extensible access control markup language,可扩展的访问控制标记语言)[3]是一种基于 XML 的开放标准语言,它为 ABAC 提供了利用属性进行访问控制的策略描述语言,并给出了基本的 ABAC 授权框架,是 ABAC 策略描述语言的理想选择[2]。但在 XACML

收稿日期: 2012-06-11; **修回日期**: 2012-08-09 **基金项目**: 广东省自然科学基金资助项目(S2011010004197);广东省教育部产学研合作项目(2011B090400331,2011B090400367)

作者简介:熊智(1978-),男,湖北黄冈人,副教授,硕导,博士,主要研究方向为信息系统安全、集群服务器(witmmx@gmail.com);王平(1986-),男,硕士研究生,主要研究方向为云计算、信息系统安全;徐江燕(1986-),女,硕士研究生,主要研究方向为访问控制、信息系统安全;蔡伟鸿(1963-),男,教授,硕导,博士,主要研究方向为访问控制、数字版权管理.

中,访问请求和访问规则都采用XML文件来描述,决策时由于需要解析这两个XML文件,因而运行开销较大。另外,访问规则文件中的XML标签很复杂,需要由专业人士编写。基于属性的加密体制[4,5]是ABAC的一种应用方式,它将数据加密和访问控制融为一体,让密文和密钥分别对应一个访问结构(规则)和一个属性集合(或反过来),只有当属性集合满足访问结构时才能解密数据。但该体制中的访问规则的表达能力很弱,只能表达实体具有或不具有某种属性,而无法将实体属性与数值或字符串等进行比较。规则引擎也可以用来描述访问规则[6],并且能表达较复杂的规则,但由于规则引擎并不是为访问控制而建立的,故其实现较复杂,运行开销较大。另外,由于规则是存储在一定格式的文本文件中,其解析开销也较大。还有一些文献(如文献[7,8]等)自己定义访问规则的形式,并自己实现规则的解析和执行,这不但实现工作量巨大,而且在规则的表达能力以及规则的解析和执行效率上也很难具有优势。

对于企业云存储的访问控制方案来说:

a)用户的每个操作都需要进行访问决策,访问规则的解析和运行开销要小;

b)访问规则可能由普通用户制定,规则的编写要很容易;

c)为了进行精细的访问控制,规则的表达能力要很强。

因此,现有的ABAC方案无法满足企业云存储的应用需求。为此,本文提出了一种满足上述需求的适合企业云存储应用的ABAC方案。此外,考虑到云存储中资源数量巨大和层次结构的特点,该方案允许继承访问规则,以简化访问规则的制定和减小访问规则的存储空间。

1 企业云存储系统的整体设计

1.1 体系结构

企业云存储系统的体系结构如图1所示。

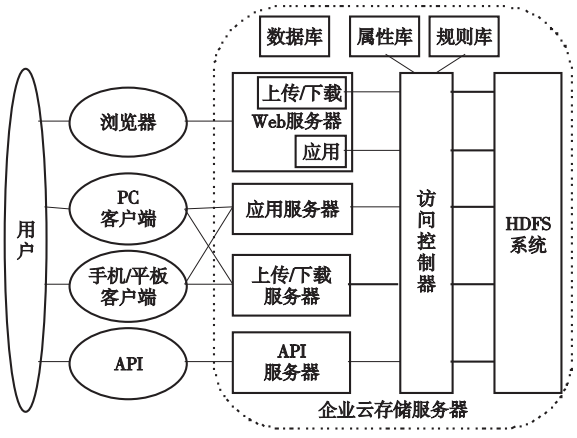

图1 企业云存储系统的体系结构

用户可以通过浏览器、PC客户端、手机/平板客户端或API来访问云存储服务器,并且用户的访问操作可分为两类:a)上/下载文件的操作;b)非上/下载文件的其他操作,如文件的删除、目录的创建等,本文将其称为应用操作。用户的任何访问操作都需要事先询问访问控制器,访问控制器则根据用户属性、被访问资源的属性、当前的环境、操作的类型以及访问规则进行决策。只有当决策结果为true时该操作才被允许执行。其中用户属性和资源属性存储在属性库中,访问规则存储在规则库中。为了保证服务性能,各种服务均采用集群服务器来提供可伸缩的服务。

1.2 企业数据的存储

系统采用HDFS(hadoop distributed file system)存储企业的数据文件。HDFS是Google GFS的开源实现,是一个高度容错的分布式文件系统,它能够提供高吞吐量的数据访问,并且可以部署在低廉的硬件上。系统采用单实例多租户的方式为大量企业提供云存储服务,每个企业对应一个英文缩写,其数据存储在"{HDFS数据目录}/{企业英文缩写}"目录下,并采取一定的数据加密措施以保证其安全。

2 实体属性及其存储

2.1 实体及实体属性

访问规则中允许使用主体(subject)、资源(resource)和环境(environment)三种实体。主体是指访问者;资源是指被访问的对象,包括文件和目录,每个资源对应一个路径;环境是指访问过程的上下文,包括客户端的类型和访问时间等。

不同类型的企业之间存在差异,它们需要用到的主体(员工)属性也有所不同,例如学校员工的属性需要包括院、系、职称等,而工厂员工的属性需要包括部门、车间、工种等,并且有些企业可能需要用到一些特殊的员工属性。因此,将主体属性分为基本主体属性和企业自定义主体属性。基本主体属性由系统制定,包括用户名、邮箱、姓名等一些通用且必需的员工属性;而企业自定义主体属性则是在企业注册时设置的。企业在注册时会连同注册一个管理员用户(admin),其拥有管理该企业数据的最高权限。企业普通用户的属性由企业的admin在创建该用户时填写,且只能由admin修改,用户自己无法修改。

资源属性和环境属性由系统制定。资源属性包括路径、类型(文件或目录)、扩展名、大小、哈希值(以上三属性仅针对文件)、修改日期、修改时间、拥有者和密级(普通、机密、绝密或继承,继承表示继承其上级资源的值);环境属性包括客户端类型、客户端IP、(服务器)负载(0~1之间的数值)、(访问)日期和时间。

2.2 属性的存储

环境属性无须存储,由系统根据当时实际情况动态生成;资源属性存储在数据库中。显然,基本主体属性和企业自定义主体属性也需要存储到数据库中。但如果采用传统的关系型数据库来存储企业自定义的主体属性,无论存储还是查询时都需要对多个表进行操作,效率较低。另外,数据库需要存储海量的用户属性和资源属性。因此,系统需要采用一个结构松散且扩展性好的数据库。

MongoDB[9]是一种开源的面向文档的数据库。一个MongoDB服务可以建立多个数据库,每个数据库可以有多个集合(相当于关系数据库中的表),每个集合可以存放多个文档(相当于关系数据库中的记录),文档被存储为键/值对的形式。MongoDB具有以下优点:

a)以文档为单位存储,易存储对象类型的数据,很适合存储实体的属性;

b)模式自由,不需要提前设计结构,非常适合存储企业自定义的主体属性,且方便属性的扩展;

c)支持的查询语言非常强大,且支持建立索引以提高查

询效率;

d)自动分片功能支持水平的数据库集群,支持云级别的伸缩性。

因此系统选用 MongoDB 作为数据库。

假设某学校的英文缩写为 abc,那么注册时需要用一个集合 abc. attr 来存储其自定义的主体属性,每个属性用一个文档表示。例如:

```
{'name':'学院','type':'枚举','值':'工学院,文学院,…'},
{'name':'职务','type':'枚举','值':'校长,院长,…,无'},
{'name':'入职日期','type':'日期'},
…
```

属性的类型(type)主要有两个作用:a)区别该属性的输入方式,如枚举类型的用选择框输入、日期类型的用日期控件输入等;b)提供检查该属性数据是否合法的依据。当增加普通用户时,则在集合 abc. staff 中增加如下文档:

```
{
  '用户名':'zhangsan',
  '邮箱':'zhangsan@ abc. edu. cn',
  '姓名':'张三',
  …  #其他基本主体属性
  '学院':'工学院',
  '职务':'系主任',
  '入职日期':'2006-07-01',
  …  #其他企业自定义主体属性
}
```

3　访问规则的描述及存储

3.1　访问权限

由于访问规则的制定者是每个企业的管理员或普通用户,如果权限的种类太多,会使得规则的制定过于复杂且容易出错,因此系统目前只定义了三种权限,即读、写和管理,它们对应的具体操作如表 1 所示。

表 1　权限对应的操作

	读(read)	写(write)	管理(manage)
目录	列举目录下的文件和目录	重命名,创建文件和子目录	删除,修改拥有者,修改密级,修改访问规则
文件	读取文件	重命名,修改文件	

3.2　访问规则的描述

每个资源的每种访问权限都对应一条访问规则。访问规则为一逻辑表达式,由主体、资源和环境三者的属性,算术、逻辑、关系等运算符,以及一些常用的函数构成。其执行结果为布尔值,当执行结果为 true 时表示允许访问,为 false 时表示禁止访问。本文采用 Python 逻辑表达式描述访问规则,并采用 eval 函数执行规则。由于 Python 语言具有灵活且强大的表达能力,加上访问规则表达式只要符合 Python 的语法,且执行结果为布尔值即可,因此访问规则的表达能力很强,可以制定出非常复杂的控制规则,且书写容易,只需掌握 Python 的基本语法即可。

在访问规则中,实体用字典变量来表示,实体属性用字典中的元素表示,元素的“键”对应属性名,元素的“值”对应属性值。由于字典变量无须(而对象类需要)事先定义结构,故当对实体属性进行扩展时,无须修改程序代码。本文还规定主体用变量 S 表示,资源用变量 R 表示,环境用变量 E 表示。以下是几条合法的访问规则实例:

规则 1　((S['部门'] == '财务部') and (S['职务'] in {'经理','副经理'})) or (S['姓名'] == '张三'),即允许财务部的经理和副经理,或者张三访问;

规则 2　(S['用户名'] == R['拥有者']) or (S['用户名'] == 'admin'),即允许资源的拥有者或者管理员访问;

规则 3　(REMatch(E['客户端 IP'],'^202 \. 192 \. 159 \. ')) and (YearSpan(E['日期'], S['入职日期']) > 2),即允许 IP 地址为 202. 192. 159. * 且入职超过 2 年的用户访问(REMatch 和 YearSpan 是本文实现的正则表达式匹配函数与计算两日期之间年份差的函数);

规则 4　(R['类型'] == '文件') and (R['大小'] < 2 ** 20) and (R['扩展名']. lower() not in ['. exe', '. bat']),即允许访问(如上传)大小不超过 1 MB 的非可执行文件。

3.3　访问规则的继承与存储

各企业的资源访问规则连同资源属性一起存储在集合{企业英文缩写}. res 中。为了便于文件的归档和分类,以及便于用户的检索,各企业文件的组织通常都是成树状的层次结构。并且考虑到:a)资源(目录或文件)的权限通常默认继承上一级资源(目录)的权限;b)拥有某个资源的读权限通常必须拥有其上级资源的读权限,而拥有某个资源的读权限不一定拥有其下级资源的读权限;c)拥有某个资源的写权限通常也拥有其下级资源的写权限,而拥有某个资源的写权限不一定拥有其上级资源的写权限;d)管理权限与写权限类似。因此,系统为每种权限定义了两个键:一个是 inherit(继承),键值为布尔量;另一个是 rule(规则),键值为逻辑表达式字符串。各资源对应的文档格式如下:

```
{
  '路径':'***',
  '类型':'文件',
  …  #其他资源属性
  '访问规则':{
      'read':{'inherit':***,rule:'***'},
      'write':{'inherit':***,rule:'***'},
      'manage':{'inherit':***,rule:'***'}
  }
}
```

资源各种权限的键值组合与该资源最终访问规则的对应关系如表 2 所示。

表 2　资源的最终访问规则

权限	inherit	rule	最终访问规则
读	true	为空	上级资源的最终访问规则(默认)
	true	非空	(上级资源的最终访问规则) and (rule 的键值)
	false	为空	true,即允许访问
	false	非空	rule 的键值
写和管理	true	为空	上级资源的最终访问规则(默认)
	true	非空	(上级资源的最终访问规则) or (rule 的键值)
	false	为空	true,即允许访问
	false	非空	rule 的键值

允许访问规则的继承,使得上一级资源的规则可被下一级资源重复利用,不仅大幅度减少了规则存储的空间,而且大幅度减少了规则编写的工作量。另外,本文将 inherit 为 true、rule 为空视为默认情况,此时无须存储其键和键值。由于该情况通常会大量出现,从而可进一步节省大量的存储空间。对于各企业的根目录,本文将其密级设置为普通,将其访问规则设置为

```
'read':{'inherit': false, 'rule': 'true'},
'write':{'inherit': false, 'rule':
    "(S['用户名'] == R['拥有者']) or
    (S['用户名'] == 'admin')"},
'manage':{'inherit':false,'rule':
    "(S['用户名'] == R['拥有者']) or
    (S['用户名'] == 'admin')"}
```

4 访问控制器

4.1 开发框架

访问控制器采用跨语言的服务开发框架 Thrift[10] 进行实现。采用该框架,开发人员无须关注 RPC(remote procedure call,远程过程调用)协议层和传输层的实现,而只需首先在 Thirft 文件中定义数据类型和服务接口;然后用 Thrift 编译器编译该文件产生相应语言(目前支持 C ++ 、Java、Python、PHP 等 10 多种主流语言)的代码文件;最后分别用相应的语言实现服务程序和客户端调用程序。Thrift 框架类似于 Web Service,但其效率和开发的方便性是 Web Service 所不能比拟的。

4.2 数据类型和服务接口的定义

访问控制服务接口为每种访问操作都定义了一个返回布尔值的访问控制服务函数。Thrift 文件的主要代码如下:

```
struct RequestParameter {
    1: string enterprise
    2: string user
    3: ClientType clienttype   //枚举类型, 其定义省略
    4: string clientip
}
service AccessControl {
  … //省略其他读写操作的访问控制服务函数
    bool DirMove(1: string oldpath, 2: string newpath,
                 3: RequestParameter parameter),
    … //省略其他管理操作的访问控制服务函数
    bool SetRule(1: string path, 2: RequestParameter parameter)
}
```

4.3 访问控制服务函数的实现

各服务函数在实现时,首先将访问操作转换为对某个(或某些)资源进行某种(或某些)权限的操作,然后调用决策函数 Decision 对操作进行决策并返回决策结果,Decision 又是通过调用递归的决策函数 RecursiveDecision 进行决策。服务函数也采用 Python 来实现。下面以 DirMove 为例说明服务函数的实现方法。

```
def DirMove (self, oldpath, newpath, parameter):
  return (Decision(oldpath, Right.MANAGE, parameter) and
    Decision(Father(newpath), Right.WRITE, parameter))
def Decision(resourcepath, righttype, parameter):
  enterprise = parameter.enterprise, user = parameter.user
  clienttype = parameter.clienttype, clientip = parameter.clientip
  S = GenerateSubject(enterprise, user)   #需读数据库
  E = GenerateEnvironment(clienttype, clientip)
  return RecursiveDecision(enterprise, resourcepath, righttype, S,
E)
def RecursiveDecision(enterprise, resourcepath, righttype, S, E):
  R = GenerateResource(enterprise, resourcepath)   #需读数据库
  (inherit, rule) = GetRule(enterprise, repath, righttype)
  try:
    if IsRoot(repath) or (not inherit):
      return eval(rule)
    if (righttype = = Right.READ):
      return (eval(rule) and RecursiveDecision(enterprise,
        Father(repath), righttype, S, E))
    else:
      return (eval(rule) or RecursiveDecision(enterprise,
        Father(repath), righttype, S, E))
except Exception,ex:
  规则不合语法或其中有未定义的属性,通知企业管理员
  return False
```

4.4 Eval 函数的安全性

由于访问规则是由用户编写的,为了防止用户在规则中执行恶意的代码,如利用__import__("os").system(cmd)执行删除系统文件和窥探系统信息的代码,系统必须对规则中使用的变量和函数进行限制。这可以借助 eval(expression[, globals[, locals]])函数的后两个可选参数做到,其中 globals 用于指定 expression 中可用的全局变量和函数,locals 则用于指定可用的局部变量和函数。本文在实现时允许规则中使用的局部变量为 S、R 和 E,全局函数为 REMatch 和 YearSpan 等实现的函数。注意,Python 内置的函数(如 abs、len 等)和内置对象的成员函数(如字符串的 lower 函数等)是不受这两个可选参数限制的,因而访问规则的表达能力能够得到保证。

5 实例及优势分析

下面将本文提出的 ABAC 方案与目前流行的三种 ABAC 方案,即基于 XACML 的方案、基于属性的加密体制以及基于规则引擎的方案。结合实例进行对比分析,假设需要实现的规则为:仅当主体的用户名等于资源的拥有者且客户端类型为浏览器时才允许访问。

5.1 访问规则的编写难度

基于属性的加密体制无法实现该规则。在本文的方案中,该规则可简单地描述为

```
(S['用户名'] == R['拥有者']) and (E['客户端类型'] == '浏览器')
```

若采用基于 XACML 的方案,规则文件的核心部分如下:

```
〈Condition FunctionId = "urn:oasis:names:tc:xacml:1.0:function:
and"〉
〈Apply unctionId =
  "urn:oasis:names:tc:xacml:1.0:function:string-equal"〉
〈Apply FunctionId =
  "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"〉
```

```
〈SubjectAttributeDesignator AttributeId = "username"
DataType = "http://www.w3.org/2001/XMLSchema#string"/〉
〈/Apply〉
〈Apply FunctionId =
"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"〉
〈ResourceAttributeDesignator AttributeId = "owner"
DataType = "http://www.w3.org/2001/XMLSchema#string" /〉
〈/Apply〉
〈/Apply〉
〈Apply FunctionId =
"urn:oasis:names:tc:xacml:1.0:function:string-equal"〉
〈Apply FunctionId =
"urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"〉
〈EnvironmentAttributeDesignator
DataType = "http://www.w3.org/2001/XMLSchema#string"
AttributeId = "clienttype"/〉
〈/Apply〉
〈AttributeValue DataType =
"http://www.w3.org/2001/XMLSchema#string"〉
Browser
〈/AttributeValue〉
〈/Apply〉
〈/Condition〉
```

可见其规则编写的难度很大,即使采用程序来生成该规则文件,代码也很复杂。Drools[11]是一个应用广泛的开源规则引擎实现,若用其描述该规则,规则文件的核心部分如下:

```
when
    $S:Subject()    //主体对象
    $R:Resource(owner == $S.username)    //资源对象
    $E:Environment(clienttype == "浏览器")    //环境对象
或者:
    $S:Subject() $R:Resource() $E:Environment()
    eval((  $S.username == $R.owner) && (  $E.clienttype ==
    "浏览器"))
then
    drools.getWorkingMemory().setGlobal("Result",
      new Boolean(true));
```

若采用后者 eval 的方式来表达规则,则规则的编写与本文的方案类似,也很容易。

5.2　访问规则的执行效率

在一台 CPU 为 Intel 酷睿 2 双核 E8400(主频 3 GHz)、内存为 3 GB 的 Windows XP 机器上,分别测试了该规则在三种方案(基于属性的加密体制除外)下的(本地)执行时间,XACML 实现采用的是 SUN 公司的 sunxacml[12],测试结果如下:Drools 约为 1 200 ms,sunxacml 约为 200 ms,本文方案约为 0.04 ms。在前两种方案中,由于分别需要解析文本文件和 XML 文件,故执行开销较大,而本文方案无须解析文件,规则直接执行,故执行开销很小。特别地,由于规则引擎并不是为访问控制而建立的,Drools 还实现了诸如对象修改、规则重复触发等其他功能,故其实现复杂,运行开销很大。此外,由于在基于属性的加密体制中访问控制和加密融为一体,故这里没有测试其执行效率。

5.3　访问规则的表达能力

基于属性的加密体制中访问规则的表达能力很弱,只能表达实体具有或不具有某种属性,而无法将实体属性与数值或字符串等进行比较;其他三种方案的访问规则中都能使用各种类型的变量,如算数运算符、关系运算符、逻辑运算符,集合操作,正则表达式等,并且可以自定义函数,从而表达能力都很强,但本文方案中规则的书写更为简洁明了。

6　结束语

目前市场上的企业云存储产品采用的访问控制模型无法满足应用需求,虽然 ABAC 是非常理想的访问控制模型,但目前已有的 ABAC 方案在访问规则的描述方面存在着一些不足,仍然无法满足应用需求,为此本文提出了一种新的 ABAC 方案,并给出了实现方法。该方案的设计与实现具有以下优点:a)采用 Python 逻辑表达式描述访问规则,并采用 eval 函数执行规则,从而使规则编写容易,表达能力强,且执行效率高;b)规则中采用字典变量表示实体,从而当扩展实体属性时无须修改程序代码;c)考虑到云存储中的资源具有层次结构的特点,并根据读、写和管理权限的特点,分别为不同的权限设计了相应的访问规则继承策略,不仅减少了规则存储的空间,而且减少了规则编写的工作量;d)采用模式自由的 MongoDB 作为属性存储的数据库,便于存储企业自定义的主体属性;e)采用 Thrift 框架实现访问控制器,从而能提供开放、跨语言、跨平台,且高效的访问控制服务。针对层次结构的资源设计通用的访问控制模型和框架是笔者下一步的工作。

参考文献:

[1] BONATTI P, SAMARATI P. A uniform framework for regulating service access and information release on the Web [J]. Journal of Computer Security, 2002, 10(3): 241-271.

[2] 王小明,付红,张立臣. 基于属性的访问控制研究进展[J]. 电子学报, 2010, 38(7): 1660-1667.

[3] MOSES T. eXtensible access control markup language(XACML) version 2.0 [EB/OL]. [2012-06-11]. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[4] VIPUL G, OMKANT P, AMIT S. Attribute-based encryption for fine grained access control of encrypted data [C]//Proc of ACM Conference on Computer and Communications Security. New York: ACM Press, 2006: 89-98.

[5] 孙国梓,董宇,李云. 基于 CP-ABE 算法的云存储数据访问控制[J]. 通信学报, 2011, 32(7): 146-152.

[6] 王辉. 基于规则引擎的访问控制研究[J]. 计算机安全, 2011(6): 37-39.

[7] 焦振海,丁二玉,骆斌. 工作流管理中基于规则策略的访问控制[J]. 计算机应用研究, 2008, 25(3): 885-887, 902.

[8] 王保义,王蓝婧. 电力信息系统中基于属性的访问控制模型的设计[J]. 电力系统自动化, 2007, 31(7): 81-84, 98.

[9] 10gen, Inc. MongoDB [EB/OL]. [2012-06-11]. http://www.mongodb.org/.

[10] The Apache Software Foundation. Apache Thrift [EB/OL]. [2012-06-11]. http://thrift.apache.org/.

[11] JBoss Community. Drools-The business logic integration platform [EB/OL]. [2012-06-11]. http://www.jboss.org/drools/.

[12] Sun Microsystems, Inc. Sun's XACML implementation [EB/OL]. [2012-06-11]. http://sunxacml.sourceforge.net/.