УДК 004.94 DOI 10.17223/2226308X/10/45

О ФРЭЙМВОРКЕ АТРИБУТНОГО УПРАВЛЕНИЯ ДОСТУПОМ ABAC ENGINE

Д. Н. Колегов, О. В. Брославский, Н. Е. Олексов

Работа посвящена программной реализации атрибутного управления доступом (ABAC) в различных компьютерных системах (КС). Предлагается следующее расширение метода управления доступом из XACML: 1) для описания и задания политики атрибутного управления доступом используется созданный специализированный язык (Domain Specific Language (DSL)) ALFAScript, основанный на языке ALFA; 2) политика на языке ALFAScript компилируется в код на языке программирования Lua; 3) при поступлении запроса субъекта на доступ к сущности строится соответствующий контекст (в терминах XACML), для которого происходит запуск кода на Lua в среде выполнения (runtime) механизма управления доступом КС; 4) результатом выполнения Lua-кода является решение о возможности предоставления субъекту доступа к сущности. Модель взаимодействия, набор программных компонент, структур данных и инструментов, позволяющих разработчику самостоятельно реализовать ABAC с использованием данного метода независимо от целевой КС, предметной области, элементов управления доступом, их состава и структуры и составляют фрэймворк ABAC Engine. Новизна предлагаемого подхода заключается в использовании языка Lua для интерпретации политики управления доступом в среде этой КС. Использование предложенного метода упрощает и ускоряет реализацию атрибутного управления доступом.

Ключевые слова: *управление доступом, ABAC, XACML, ALFA, Lua.*

Управление доступом является одним из основных механизмов безопасности компьютерных систем. Механизмы управления доступом реализуются не только в операционных системах (ОС) и системах управления базами данных (СУБД), но и в веб-приложениях, межсетевых экранах прикладного уровня, ERP-системах, прикладном программном обеспечении и т. п. При этом на практике часто возникает ситуация, когда разработчик, реализовав корректный механизм управления доступом на одном языке программирования, вынужден разрабатывать тот же самый механизм на другом языке программирования. Данная задача возникает, например, в случае прототипирования программного продукта на языке Python с последующим его переписыванием на языке C++ или в случае, когда механизм управления доступом был реализован для защиты СУБД, а затем его необходимо адаптировать для защиты веб-приложений [1, 2].

С точки зрения программной разработки хотелось бы иметь средство, удовлетворяющее следующим требованиям:

1) политика безопасности управления доступом в КС должна описываться на DSL-языке, позволяющем использовать типовые элементы управления доступом (например, роли, привилегии, права и т. д.);
2) политика и механизм управления доступом не должны быть привязаны к предметной области конкретной защищаемой КС (например, к веб-приложению, СУБД, ОС);
3) реализация механизма управления доступом не должна быть жёстко привязана к среде выполнения и языку программирования.

Принципы атрибутного управления доступом [3] наиболее близко соответствуют данным требованиям. Во-первых, атрибутное управление доступом отличается от дру-

гих видов управления доступом тем, что не предоставляет какую-то предопределённую политику, как это принято, например, в ролевом или мандатном управлении доступом. В случае использования ABAC администратор или разработчик должен сам разработать такую политику в соответствии с предметной областью КС, а затем описать её на каком-либо DSL (например, в XACML для этого используется язык ALFA). Во-вторых, атрибутное управление доступом является наиболее общим и с некоторыми ограничениями может быть использовано для реализации других видов управления доступом (например, ролевого, дискреционного, мандатного) [4]. Видимо, оно не может быть использовано лишь для обеспечения безопасности информационных потоков в системах с мандатным управлением доступом.

В принципе, использование XACML [5] для реализации ABAC [3] позволяет, несмотря на наличие некоторых недостатков, выполнить первые два требования. Для выполнения всех требований предлагается модифицировать метод, используемый в XACML.

Далее предполагается, что читатель знаком с основными положениями стандарта XACML и с такими его функциональными элементами, как Policy Administration Point (PAP), Policy Enforcement Point (PEP), Policy Decision Point (PDP) и Policy Information Point (PIP).

Предлагается следующая модификация XACML, реализованная в разрабатываемом фреймворке Angine (ABAC Engine).

На уровне PAP выполняются следующие действия:

1) фиксируется целевая КС, для которой необходимо реализовать механизм управления доступом (например, СУБД MySQL);
2) фиксируется целевой язык программирования, на котором необходимо реализовать механизм управления доступом (например, Python);
3) все программные элементы управления доступом защищаемой КС (сущности, объекты, субъекты, атрибуты) и отношения между ними описываются на языке описания интерфейсов (IDL);
4) с помощью разработанного генератора кода выполняется интерпретация интерфейсов, заданных на IDL, и создание соответствующих подходящих структур данных (классов, типов, объектов) для выбранного языка программирования;
5) политика безопасности описывается на специально разработанном языке ALFAScript, основанном на XACML-совместимом языке ALFA [6]. При этом из *ALFAScript* возможен доступ ко всем атрибутам субъектов и сущностей доступа;
6) с помощью разработанного компилятора выполняется преобразование кода на языке ALFAScript в код на языке Lua, доступного для PDP.

На уровне PEP выполняются следующие действия:

1) синтаксический анализ запроса и создание структур данных, соответствующих запросу на доступ субъекта к сущности (Request, RequestContext). При необходимости выполняется получение атрибутов от PIP;
2) RequestContext отправляется PDP;
3) получение ResponseContext от PDP. Запрещение или разрешение запроса в соответствии с результатом проверки запроса.

На уровне PDP выполняются следующие действия:

1) создание среды выполнения кода на Lua средствами заданного языка программирования;

2) запуск и выполнение Lua-кода на RequestContext;
3) формирование контекста ответа ResponseContext и отправка его PEP.

На уровне PIP выполняются следующие действия:

1) разрешение динамических атрибутов (например, IP-адрес, время);
2) получение значений статических атрибутов (например, уровень доступа, идентификатор).

Основной особенностью данного подхода является компиляция политики безопасности на ALFAScript в код на языке Lua. В известных фреймворках и библиотеках по политике управления доступом, выраженной на каком-либо языке, генерируется или XML-код, или код для целевого языка программирования (например, [7]), а затем данный код выполняется в КС. Выбор языка Lua в качестве универсальной платформы выполнения обосновывается тем, что в настоящее время Lua позиционируется и используется как встраиваемый язык. Все основные языки программирования (например, C++, Java, GoLang, Python, Ruby, NodeJS) имеют развитые средства для запуска программ на Lua и взаимодействия с ними через среду выполнения.

В рамках создаваемого фреймворка разработаны и реализованы следующие средства:

— прототип языка описания универсальных синтаксических деревьев (UST);
— прототип языка ALFAScript;
— компилятор политики безопасности из ALFAScript в Lua;
— генератор структур данных для языков C++, Python и Go;
— программные средства, реализующие функции PAP, PEP, PIP и PDP [5] для языков Python и Go;
— синтаксические анализаторы MySQL, TSQL, HTTP для PEP.

Рассмотрим пример. Пусть необходимо реализовать управление доступом на уровне межсетевого экрана для СУБД MySQL [1, 2]. Определим следующие интерфейсы элементов управления доступом:

```
interface Entity {
  abstract id: String;
}

interface SQLEntity <: Entity {
  database: String;
  table: String;
  column: String;
  level: Number;
  tags: [String];
}

interface Subject <: Entity {
  name: String;
  level: Number;
  tags: [String];
}
```

Данные интерфейсы могут быть реализованы следующими классами на языке Python:

```
1
2  from abc import ABCMeta, abstractproperty, abstractmethod
3
4  class Entity(object):
5      def __init__(self):
6          super(Entity, self).__init__()
7
8      @abstractproperty
9      def id(self):
10         pass
11
12
13 class SQLEntity(Entity):
14     def __init__(self, database=None, table=None, column=
       None, level=None, tags=None):
15         super(SQLEntity, self).__init__()
16         self.database = database
17         self.table = table
18         self.column = column
19         self.level = level
20         self.tags = tags
21
22
23 class Subject(Entity):
24     def __init__(self, name=None, level=None, tags=None):
25         super(Subject, self).__init__()
26         self.name = name
27         self.level = level
28         self.tags = tags
```

Пример политики безопасности на языке ALFAScript:

```
1  namespace example {
2    export policy Main {
3      target clause action == "select"
4      apply denyUnlessPermit
5      rule r1 {
6        permit
7        target clause subject.level > entity.level
8      }
9    }
10 }
```

Код на языке Lua, реализующий приведённую выше политику безопасности:

```
1  local lib = require('angine/lib/alfa')
2  local __test = lib.test
3  local __subseteq = lib.subseteq
4  local __issubseteq = lib.issubseteq
5  local __iselement = lib.iselement
6
7  local example = {}
8
```

```
local __example = function(example)

  function example.Main(ctx, actions, handlers)
    if not ctx.action then
      return actions.indeterminate
    end
    if not ( ctx.action == "select" ) then
      return actions.notapplicable
    end


    local function r1(ctx, actions, handlers)
      if not ctx.entity.level or not ctx.subject.level then
        return actions.indeterminate
      end
      if ( ctx.subject.level > ctx.entity.level ) then
        return actions.permit
      end
      return actions.notapplicable
    end

    local rules = { r0 = r1 }
    for _, rule in pairs(rules) do
      local decision = rule(ctx, actions, handlers)
      if decision == actions.permit then
        return actions.permit
      end
    end
    return actions.deny

    end

end

__example(example)

function __main(ctx, actions, handlers)
  return example.Main(ctx, actions, handlers)
end
```

Пример итогового кода, реализующего управление доступом (PEP):

```
from Angine.policy import Policy
from Angine.pip import PIP
from Angine.pdp import PDP

alfa_mysql_policy = """
    namespace example {
        export policy Main {
          target clause action == "select"
            apply denyUnlessPermit
            rule r1 {
```

```
                permit
                target clause subject.level > entity.level
            }
        }
    }
"""

def pep():
    ...
    request = get_request(network)
        policy = Policy(alfa_mysql_policy)
    pip = PIP.init_data(mongo_connection)
    pdp = PDP(policy.get_lua_policy())
    ctx = pip.create_ctx(request)
    response = pdp.evaluate(ctx)
    if response["result"]["decision"] != "permit":
        return None
        else:
            return process(request)
```

Таким образом, в работе представлено расширение метода управления доступом из XACML — фрэймворк атрибутного управления доступом Angine, позволяющий гибко реализовывать механизм управления доступом в различных КС. Принципиально новым является компиляция заданной политики управления доступом на языке ALFAScript в код на языке Lua с последующим его запуском в среде выполнения языка программирования.

ЛИТЕРАТУРА

1. *Колегов Д. Н., Ткаченко Н. О.* Неинвазивная реализация мандатного управления доступом в веб-приложениях на уровне СУБД // Прикладная дискретная математика. Приложение. 2015. № 8. С. 89–92.
2. *Колегов Д. Н., Ткаченко Н. О.* Неинвазивное устранение уязвимостей логического управления доступом в веб-приложениях [Электронный ресурс]. Режим доступа: `https://www.youtube.com/watch?v=SPiY6D3M0yE`
3. *Hu V. C., Ferraiolo D., Kuhn R., et al.* Guide to Attribute Based Access Control (ABAC) Definition and Considerations [Электронный ресурс]. Режим доступа: `http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf`
4. *Jin X., Krishnan R., and Sandhu R.* A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. [Электронный ресурс]. Режим доступа: `http://www.profsandhu.com/confrnc/ifip/ABAC-dbsec12-preprint.pdf`
5. *OASIS Standard.* eXtensible Access Control Markup Language (XACML) Version 3.0. [Электронный ресурс]. Режим доступа: `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`
6. *Axiomatics.* How Can I Use Policy References in ALFA? [Электронный ресурс]. Режим доступа: `https://www.axiomatics.com/blog/how-can-i-use-policy-references-in-alfa/`
7. *Decat M.* STAPL. The Simple Tree-structured Attribute-based Policy Language. [Электронный ресурс]. Режим доступа: `https://github.com/stapl-dsl`