sqlmap [4], некоторые сигнатуры добавлены авторами. Сигнатуры хранятся в формате JSON. Пример сигнатуры рализованного модуля BeEF:

```
1 {
2   "name":"F5 BIG-IP ASM",
3   "cookie": ["TS[a-zA-Z0-9]{3,8}"],
4   "headers": []
5 }
```

ЛИТЕРАТУРА

1. *Alkorn W., Frichot C., and Orru M.* The Browser Hacker's Handbook. Indianapolis, John & Wiley Sons, 2014. 648 p.
2. The Browser Exploitation Framework Project. `http://beefproject.com/`
3. The WAFW00F project. `https://github.com/EnableSecurity/wafw00f`
4. The sqlmap project. `http://sqlmap.org/`

УДК 004.94 DOI 10.17223/2226308X/9/36

ЛЕГКОВЕСНАЯ РЕАЛИЗАЦИЯ МЕХАНИЗМА АТРИБУТНОГО УПРАВЛЕНИЯ ДОСТУПОМ ДЛЯ СУБД НА УРОВНЕ ЗАЩИТНОГО ЭКРАНА

Д. Н. Колегов, Н. О. Ткаченко

Рассматривается легковесная реализация механизмов атрибутного управления доступом для систем управления базами данных на уровне защитных экранов приложений (*Web Application Firewall*, *Database Firewall*), функционирующих в режиме прокси-сервера. Предлагается механизм проекции ролей для добавления элементов ролевого управления доступом.

Ключевые слова: *управление доступом, ABAC, RBAC, защитный экран, СУБД.*

В настоящее время политики безопасности для приложений являются, как правило, ролевыми (*RBAC*), атрибутными (*ABAC*) или гибридными, сочетающими в себе как ролевое, так и атрибутное управление доступом. Для каждого управления доступом в отдельности имеются или создаются детально проработанные стандарты [1, 2], но они ориентированы на реализацию в конечных системах, а не в защитных экранах приложений, а потому не могут быть использованы без существенной адаптации как к условиям функционирования защитных экранов в режиме прокси-сервера, так и к самим защищаемым системам управления базами данных (СУБД).

Задача ставится следующим образом. Имеется СУБД и заданная политика безопасности, которая не может быть реализована встроенными механизмами управления доступом этой СУБД. Необходимо реализовать эту политику безопасности без изменения конфигурации, исходного кода и данных на СУБД. Такая реализация называется неинвазивной и была предложена авторами в работах [3, 4]. Для решения поставленной задачи строится легковесный механизм атрибутно-ролевого управления доступом — *ABAC-lite*.

Реализуемый механизм состоит из следующих структурных элементов в терминологии *NIST ABAC* [1]. *Policy Enforcement Point* (*PEP*) отвечает за получение и парсинг *SQL*-запросов от клиента к серверу СУБД и формирование запросов средствами

протокола HTTP к *PDP*. *Policy Decision Point* (*PDP*) вычисляет решение о возможности получения доступа субъекта к сущности. *Policy Information Point* (*PIP*) реализует получение всех атрибутов субъектов и сущностей для принятия решения *PDP*. *Policy Administration Point* (*PAP*) отвечает за администрирование, представлления и хранения политики безопасности. В рамках реализации получены и используются следующие новые результаты.

Для связи атрибутного и ролевого управления доступом впервые используется механизм проекций. Пусть $e_1, \ldots, e_M$ — сущности, $r_1, \ldots, r_N$ — права доступа, $role_1, \ldots, role_K$ — роли, где $role_k = \{(e_i, r_j)\}$ для некоторых $i \in \{1, \ldots, M\}$, $j \in \{1, \ldots, N\}$, $k \in \{1, \ldots, K\}$. Проекцией ролей на сущность e_i будем называть набор вида $pr(e_i) = \{(role_k, r_j) : (e_i, r_j) \in role_k\}$. Данный механизм позволяет более легко реализовать гибридное атрибутное и ролевое управление доступом.

Политика безопасности представляет собой набор правил, составленных на специальном языке *AF Rules*, ориентированном на применение в графических пользовательских интерфейсах и представленном в нотации *JSON*. Правило состоит из условия и решения, которое применяется при выполненном условии. Правила проверяются в порядке их записи в политике. Условие правила может содержать атрибуты объектов Сессии (*session*), Пользователя (*user*), Ресурса (*resource*) и Окружения (*env*), логичесие операторы, операторы сравнения. Возможны два вида решения: разрешить или запретить. Последнее правило является правилом по умолчанию, условие данного правила всегда выполняется. Существует также возможность сконфигурировать способ выбора решения в политике [5]:

1) *firstApplicable* — будет выбрано первое решение из правила, условия которого выполнены;
2) *permitOverrides* — будет выбрано первое «разрешающее» правило. В отсутствие «разрешающих» правил или в случае невыполнения их условий возвращается решение по умолчанию;
3) *denyOverrides* — будет выбрано первое «запрещающее» правило. В отсутствие «запрещающих» правил или в случае невыполнения их условий возвращается решение по умолчанию.

В примере, приведённом ниже, проверяется наличие группы у пользователя, и если такая задана, то группа пользователя сравнивается с группой ресурса. При этом сравнение предполагает использование иерархии групп, которая описывается отдельно в виде дерева. Если группы не сравнимы с точки зрения заданной иерархии, то условие не выполняется. Если группа пользователя ниже по иерархии группы ресурса, то условие не выполняется. В остальных случаях последняя часть условия будет выполнена.

```
1 {
2  "and": [
3   {"session.groups>isSet": true},
4   {"session.groups>any": {"in":"resource.groups>rbac.hge"}}
5  ],
6     "decision":"permit"
7 }
```

В множество атрибутов, помимо атрибутов, свойственных сущностям СУБД (базы данных, таблицы, столбцы и т. д.), входят: уровень доступа, уровень конфиденциальности, роли, группы, тэги. Уровни доступа и конфиденциальности задаются числами.

В правиле может быть описан способ сравнения уровней, что позволяет реализовать управление доступом на основе меток. Группы — иерархическая структура, которая позволяет отразить внутреннюю структуру организации. Тэги позволяют определить принадлежность субъекта к какому-либо отделу внутри организации. С точки зрения ролевого механизма наиболее интересен способ задания ролей.

Для создания и назначения роли необходимо конфигурируемой сущности присвоить имя роли и привилегии. Пример конфигурации для таблицы *test* из базы данных *test_db* приведён ниже; здесь роль с именем *gen_dir* разрешает доступы типа *select*,*insert*,*delete* и *update* к сущности *test_dbtest*.

```
{
    "id":{
        "database":"test_db",
        "table":"test"
    },
    "level": 3,
    "tags": ["accounting"],
    "privileges":[
        {
           "name":"gen_dir",
           "rights":["select","insert", "delete", "update"]
        }
    ]
}
```

Для использования политики *PAP* в механизме управления доступом правила транслируются из формата *JSON* в код на языке *Python* с помощью разработанного транслятора. Сгенерированный модуль содержит функцию *checkAccess*, которая применяет все проверки, описанные в правилах политики, и в случае выполнения условий возвращает решение о предоставлении доступа.

Таким образом, в работе представлены результаты очередного этапа разработки легковесной системы ролевого и атрибутного управления доступом, ориентированной на применение в защитных экранах приложений. В системе разработаны новые механизмы: 1) проекции ролей, 2) проверки доступа по иерархии разрешающих и запрещающих ролей генерации, 3) генерации кода на основе политики безопасности.

ЛИТЕРАТУРА

1. *Hu V. C., Ferraiolo D., Kuhn R., et al.* Guide to Attribute Based Access Control (ABAC) Definition and Considerations [Электронный ресурс]. http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf
2. Role Based Access Control. American National Standarts Institute, Inc., 2004. http://profsandhu.com/journals/tissec/ANSI+INCITS+359-2004.pdf
3. *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.
4. *Колегов Д. Н., Ткаченко Н. О.* Неинвазивное устранение уязвимостей логического управления доступом в веб-приложениях [Электронный ресурс]. https://www.youtube.com/watch?v=SPiY6D3M0yE
5. *Brossard D.* Understanding XACML combining algorithms [Электронный ресурс]. https://www.axiomatics.com/blog/entry/understanding-xacml-combining-algorithms.html