Модуль управление доступом на основе атрибутов для веб-запросов из разных источников

Т.В. Ульби, О.Р. Лапонина

***Аннотация* – В статье проанализированы возможности стандартов RBAC и ABAC для управления доступом к ресурсам из различных источников.**

К достоинствам ABAC относится то, что он позволяет описать бесконечное количество различных сценариев без необходимости создавать новые роли, что позволяет более гибко создавать и изменять правила. В статье выделены следующие преимущества ABAC: возможность описывать управление доступом в терминах, близких к терминам бизнес-логики приложения, возможность существенно автоматизировать создание как простых, так и сложных правил управления доступом, в том числе правил с динамическими параметрами. Для описания правил управления доступом используется язык XACML.

В настоящее время для управления доступом к веб-ресурсам из разных источников используется стандарт CORS.

В статье рассмотрена архитектура модуля для среды Node.js, предназначенного для управления доступом на основе атрибутов для кросс-доменных источников. Модуль включает базовые инструменты для настройки ABAC и CORS в Node.js приложениях. Модуль предоставляет веб-интерфейс, с помощью которого администратор сервиса может в режиме реального мнения изменять набор правил для различных точек доступа (Endpoints) приложения. Модуль удовлетворяет следующим функциональным требованиям: веб-интерфейс для настройки правил доступа; декларативная настройка кросс-доменных запросов для различных источников (CORS); настройка доступа на основе атрибутов как ко всему приложению, так и к отдельным точкам доступа. Приведено описание работы модуля в соответствии с XACML.

***Ключевые слова* – RBAC, ABAC, CORS, XACML, PDP, PAP, PEP, Access Control, Node js.**

I. Введение

Веб-приложения вытеснили традиционные настольные приложения практически для всего, что требует сетевого взаимодействия, и становятся конкурентоспособными в других областях. Современные веб-приложения позволяют работать с очень важными данными и выполнять серьезные операции, в том числе финансовые. Эти факторы приводят к постоянным попыткам похитить данные или деньги пользователей, нанести вред, уничтожить инфраструктуру подобных систем. Для предотвращения подобных действий специалисты в области информационной безопасности и программной инженерии постоянно разрабатывают новые механизмы защиты, подходы к разрешению доступа одним категориям пользователей и способы запрета другим.

Одной из важнейших задач в современных веб-приложениях является процесс аутентификации и авторизации пользователей [1], а также управление доступом к тем или иным ресурсам для различных групп пользователей.

На текущий момент существует большое количество подходов по организации управления доступом, но самые часто используемые – RBAC (Управление доступом на основе ролей) и ABAC (Управление доступом на основе атрибутов). Оба подхода имеют как свои преимущества, так и свои недостатки. Подробнее о них в следующих главах.

Помимо этого, важным элементом приложений, взаимодействующих по сети, является управление доступом к ресурсам между различными источниками. В современных браузерах предусмотрен стандарт CORS, контролирующий взаимодействие между ресурсами. CORS работает поверх HTTP заголовков и настраивается на веб-сервере.

II. Анализ существующих способов управления доступом к ресурсам

A. Управление доступом на основе ролей

RBAC (Role-based access control) – подход к управлению доступом на основе ролей [2]. Каждому пользователю в системе при регистрации определяется роль, а для роли определены разрешения доступа (permission).

Ролевое разграничение доступа позволяет реализовать гибкие, изменяющиеся динамически в процессе функционирования компьютерной системы правила разграничения доступа. Данный подход часто используется в системах, для пользователей которых чётко определён круг их должностных полномочий и обязанностей. При этом роли могут образовывать сложные иерархические структуры, благодаря чему наследуемая роль может перенять все свойства родительской [7].

Существует несколько моделей RBAC. В эталонной модели RBAC определен набор компонентов RBAC (пользователи, роли, разрешения, операции и объекты) и отношений, содержащих подмножества декартовых произведений, означающих реальные назначения) и множества функций отображения, которые определяют отображение экземпляров одного множества на другое. Разрешение означает возможность выполнения операции для одного или более защищаемых RBAC объектов. Типы операций и объектов, которыми

Статья получена 20 апреля 2023.
Т.В. Ульби - МГУ имени М.В. Ломоносова (email: akva_ten@mail.ru)
О.Р. Лапонина - МГУ имени М.В. Ломоносова (email: laponina@oit.cmc.msu.ru)

управляет RBAC, зависят от типа системы, в которой они реализованы.

Центральным понятием RBAC является понятие отношений ролей, в результате чего роль является семантической конструкцией, вокруг которой формулируется политика.

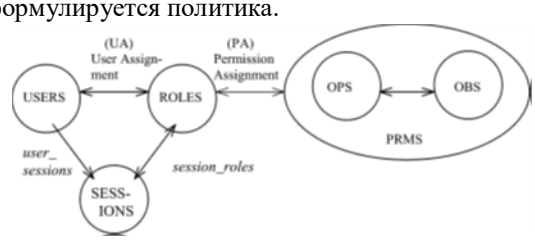

Рис. 1. Базовый RBAC

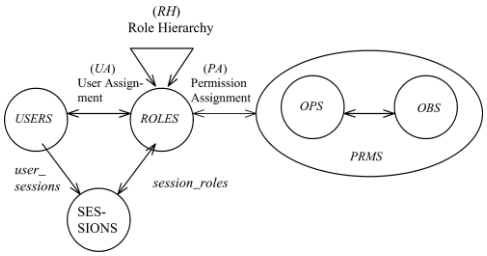

Рис. 2. Иерархический RBAC

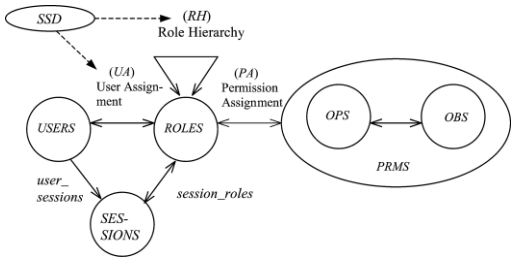

Рис. 3. Иерархический RBAC со статическим разделением ответственности

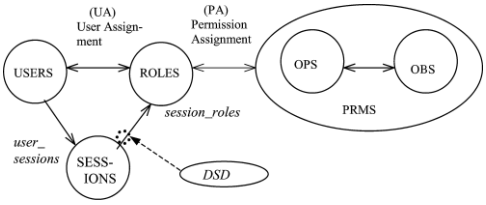

Рис. 4. Динамическое разграничение ответственности

Пример выдачи доступов для различных операций (чтение, удаление, создание, редактирование) представлен на рисунке 5.

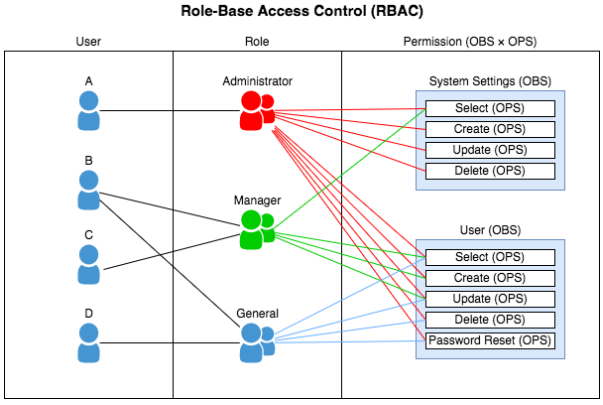

Рис. 5. Взаимосвязь пользователей, ролей и разрешений доступа при использовании RBAC

Если бизнес-правила одномерны, и все действия можно разбить по ролям (бухгалтер, менеджер, администратор и т. п.), такого подхода будет достаточно. Тогда одному бизнес-правилу будет соответствовать одна роль. Но часто правила усложняются и становятся распределенными. Чтобы продолжить взаимодействовать с системой приходится создавать новые роли.

Пример:

1) Юрист должен иметь доступ к документообороту системы. Создается роль «юрист».
2) У организации появляется филиал и юристам необходимо выдать доступы к документообороту филиала. Создается роль «юрист.филиал А».

При этом роль «Юрист.филиал А» может наследовать все свойства и правила от роли «Юрист», что позволяет избегать дублирования в описании бизнес-правил. Пример иерархической структуры ролей представлен на рисунке 6.

Рис. 6. Иерархическая структура ролей

При этом каждый субъект в системе может обладать несколькими ролями, что позволяет создавать различные комбинации прав доступа для одного пользователя и при необходимости эти права можно добавлять или удалять.

Достоинства и недостатки RBAC

Подход определения прав доступа на основе ролей является интуитивно понятным для человека и обладает рядом достоинств:

1. Легко выдавать и забирать права доступа у пользователей системы.
2. Очень легко описывать одномерные бизнес-правила без сложных условных логических конструкций.
3. За счет иерархической структуры с наследованием бизнес-правил легко создавать новые роли и расширять существующие правила [5].
4. Легко поддерживать программный код, если количество ролей в системе строго ограничено, например, должностями в организации.

RBAC зарекомендовал себя как надежный, проверенный временем инструмент, однако и при его использовании возникают определенные сложности и недостатки.

1. Тяжело поддерживать сложные многомерные бизнес-правила. Для каждого нюанса приходится создавать отдельную роль.
2. Невозможно поддерживать правила с динамическими параметрами.

3. Иерархическая структура является как преимуществом, так и недостатком одновременно. При большом количестве ролей тяжело вносить изменения, так как вся цепочка дочерних ролей будет наследовать свойства родителя, что может привести к неожиданным последствиям.

B. Управление доступом на основе атрибутов

ABAC (Attribute-based access control) – политика определения прав доступа на основе атрибутов позволяет описать бесконечное количество различных сценариев без необходимости создавать новые роли, что позволяет более гибко создавать и изменять правила [3].

Можно выделить несколько категорий атрибутов:

1. Атрибуты ресурса – ресурс, над которым проводится действие.
2. Атрибуты субъекта – права доступа текущего пользователя.
3. Атрибуты действия - создание, редактирование, удаление, чтение ресурсов.
4. Атрибуты среды – IP-адрес, устройство, версия браузера и т.д. [10].

Рассмотрим пример, когда необходимо разрешить доступ к списку документов всем менеджерам, стоимость заказа в которых превышает 5000 руб., а количество товаров не превышает 10.

Пример правила:

```
Субъект.должность === 'менеджер' &&
Ресурс.Стоимость >= 5000 && Ресурс.Количество <
=10.
```

За счет атрибутов мы можем выстраивать сколь угодно сложные цепочки правил и гибко их изменять напрямую из системы, если это потребуется.

Для выполнения авторизации значения всех атрибутов берутся в момент проверки прав и сравниваются с требуемыми значениями. Выполнение всех условий обеспечивает доступ к ресурсу.

Достоинства и недостатки ABAC

За счет гибкого описания правил ABAC позволяет избежать проблем, которые имеются в RBAC. Из достоинств можно выделить следующие:

1. Бизнес-правило не привязано к конкретной роли и его легко изменить.
2. Отсутствие иерархических правил позволяет упростить систему, и при этом сохраняется возможность переиспользовать правила. [6]
3. Отсутствуют ограничения в сложности бизнес-правил, их можно описывать по месту и при этом нет необходимости в создании новых ролей.
4. Легко задавать правила с динамически изменяемыми атрибутами, что позволяет определять их без изменений в программном коде.

ABAC как и любой другой более гибкий и динамический инструмент также обладает и своими недостатками:

1. Для удобного использования требуется определенная инфраструктура.
2. Избыточен для простых систем с прозрачными бизнес-правилами.

C. Сравнение ABAC и RBAC

Сравнение двух подходов [4], рассмотренных в данной статье представлено в таблице 1.

Таблица 1. Сравнение функциональных возможностей RBAC и ABAC

Функциональная возможность	RBAC	ABAC
Описание, близкое к терминам бизнес-логики	Нет	Да
Без лишнего ручного труда	Нет	Да
Поддержка простых правил	Да	Да
Поддержка сложных правил	Да	Да
Поддержка правил с динамически параметрами	Нет	Да
Фильтрация данных	Нет	Да

D. Управление доступом к веб-ресурсам для запросов из сторонних источников

CORS (Cross-Origin Resource Sharing) определяет механизм, который позволяет ограничить доступ к ресурсам по протоколу HTTP, если запрос сделан из другого домена, внешнего по отношению к домену, на котором расположен запрашиваемый ресурс.

Это означает, что, если приложение xyz.com делает запрос к something.io, используя API либо XMLHttpRequest, либо fetch, при использовании CORS следует включать HTTP-заголовки, которые информируют приложение, имеет ли xyz.com права доступа к something.io. Права доступа должны быть сконфигурированы на something.io, где следует указать, какие методы и из какого источника могут быть выполнены.

CORS также описывает протокол, с помощью которого браузеры делают «предварительный» запрос на сервер, на котором размещен ресурс, чтобы убедиться, что сервер разрешит фактический запрос. В этой предварительной проверке браузер отправляет заголовки, указывающие метод HTTP, и заголовки, которые будут использоваться в фактическом запросе [11].

Источники считаются разными, если у них отличается хотя бы один из параметров:

1. Порт,
2. Протокол, причем http и https считаются разными протоколами
3. Хост, например, domain.com.

Таблица 2. Принципы определения единого источника

URL	Решение	Причина
https:domain:5000.com/articles	Тот же источник	Совпадает порт, адрес, протокол
http:domain:5000.com/articles	Междоменный доступ	Отличается протокол
https:domain.com	Cross-origin	Отличается порт
https:domain.5000.ru	Cross-origin	Отличается адрес (домен верхнего уровня)

В целях безопасности браузеры ограничивают cross-origin запросы, инициируемые скриптами. Запросы между различными источниками могут приводить к различным атакам, например, межсайтовому выполнению сценариев (XSS).

Стандарт CORS определяет новые HTTP-заголовки, которые позволяют серверам описывать набор источников, которым разрешено получать доступ к информации, запрашиваемой веб-браузером. В частности, для методов HTTP-запросов [12], которые могут привести к побочным эффектам над данными сервера (HTTP-методы, отличные от GET или для POST запросов, использующих определённые MIME-типы), спецификация требует, чтобы браузеры "предварительно проверяли" запрос, запрашивая поддерживающие методы с сервера с помощью метода HTTP-запроса OPTIONS и затем, после "подтверждения" с сервера, отсылали фактический запрос с фактическим методом HTTP-запроса.

Стандарт CORS делит запросы на простые и сложные. Простые запросы не требуют дополнительных проверок, если удовлетворяют следующим условиям:

Допустимые методы для запроса:

- GET
- HEAD
- POST

Использование любых заголовков, кроме тех, что автоматически проставляются user-agent'ом (например, Connection, User-Agent, или любой другой заголовок с именем, определённым в спецификации метода Fetch в секции “Запрещённые имена заголовков (которые нельзя изменить программно)”), допустимыми заголовками, которые могут быть проставлены вручную, являются те заголовки, которые определены спецификацией метода Fetch как “CORS-безопасные заголовки запроса”, такие как:

- Accept
- Accept-Language
- Content-Language
- Content-Type – при условии, что используется одно из следующих значений: application/x-www-form-urlencoded, multipart/form-data, text/plain.

В отличии от простых запросов, сложные preflight запросы сначала отправляют HTTP-запрос методом OPTIONS к ресурсу на другом домене, чтобы определить, является ли фактический запрос безопасным для отправки. Cross-origin запросы предварительно просматриваются таким образом, так как они могут быть причастны к пользовательским данным.

Сервер должен добавить в ответ на OPTIONS запрос соответствующие заголовки, если считает источник безопасным.

Например, подобный заголовок Access-Control-Allow-Origin: http://domain.com разрешает доступ для источника http://domain.com и добавляет его в whitelist на определенный промежуток времени.

Схема взаимодействия клиента и сервера при сложном запросе представлена на рисунке 7.

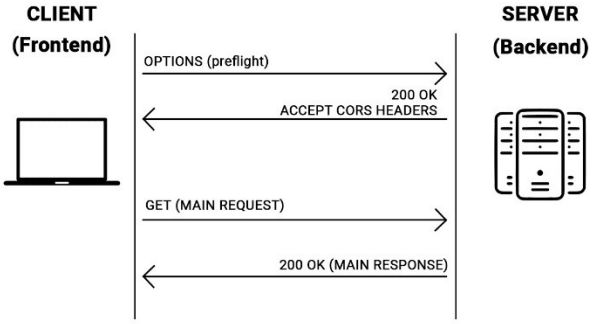

Рис. 7. Протокол взаимодействия клиента и сервера при сложном запросе

E. Язык описания политик управления доступом

XACML (eXtensible Access Control Markup Language) — стандарт, разработанный OASIS, определяющий модель и язык описания политик управления доступом, основанный на языке XML, и способы их обработки [9].

Одной из целей XACML является продвижение общей терминологии и функциональной совместимости между реализациями управления доступом нескольких разработчиков. XACML — это стандарт разграничения доступа на основе атрибутов (ABAC), где атрибуты, связанные с пользователем, действием или ресурсом, являются входными данными для принятия решения о том, может ли данный пользователь получить доступ к данному ресурсу определенным образом. Управление доступом на основе ролей (RBAC) [8] также может быть реализовано в XACML как специализация ABAC.

Основными компонентами языковой модели являются правило, политика и набор политик.

1. Правило является простейшей единицей модели. Правило должно быть включено в политику, самостоятельной единицей оно не является.

Основные компоненты правила:

Цель - определяет запросы, к которым данное правило применимо, в форме логических выражений над атрибутами запроса.

Условие - Условия представляют собой расширенную форму цели, которые могут использовать более широкий диапазон функций и, что более важно, могут использоваться для сравнения двух или более атрибутов вместе, например, subject-id == doctor-id (идентификатор объекта равен идентификатору врача).

Эффект может принимать два значения: "разрешить" или "запретить". Эффект правила срабатывает в случае положительного вычисления правила.

Обязательство - описывает то, что должно быть выполнено до или после подтверждения доступа.

Пример представлен в листинге 1.

Листинг 1. Пример правила XACML

```
  Allow access to resource MedicalJournal with
attribute patientID=x
             if Subject match
DesignatedDoctorOfPatient
             and action is read
       with obligation
            on Permit: doLog_Inform(patientID,
Subject, time)
```

```
          on Deny   :
doLog_UnauthorizedLogin(patientID, Subject, time)
```

2. Политика используется для объединения правил.

Основные компоненты политики:

- Цель
- Алгоритм объединения правил
- Набор правил
- Обязательство
- Рекомендация

Алгоритм объединения правил используется для разрешения конфликтов. Например, если одно из правил принимает значение "истина", а другое "ложь", алгоритм объединения правил определяет, какое значение примет сама политика.

3. Набор политик нужен для объединения группы политик с целью их более быстрой фильтрации на основе общего назначения, задаваемого в цели группы политик.

Основные компоненты:

- Цель
- Алгоритм объединения политик
- Набор политик
- Обязательство
- Рекомендация

Компоненты, описанные выше представлены на рисунке 8.

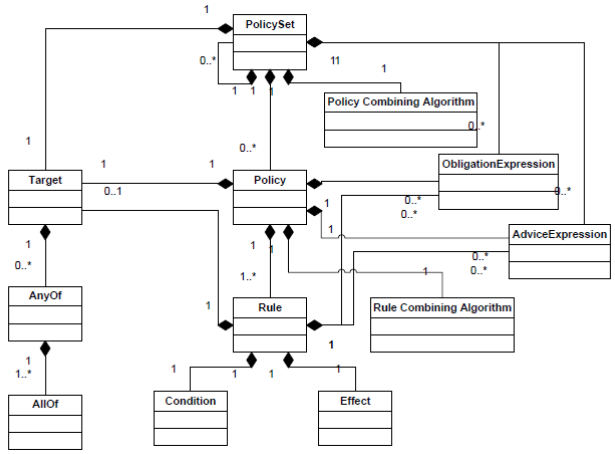

Рис. 8. Компоненты стандарта XACML

На рисунке 4 набором политик является компонент PolicySet, политика – Policy, правило - Rule. Набор политик имеет отношение один-ко-многим по отношению к политике, а политика в свою очередь такое же отношение к правилам.

Существуют определенные механизмы, которые по стандарту обрабатывают политики и правила. Схема взаимодействия этих механизмов представлена на рисунке 9.

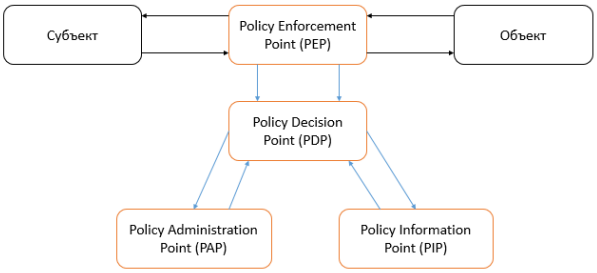

Рис. 9. Механизмы стандарта XACML

Механизм вычисления политик является центральным компонентом системы и в стандарте именуется Policy Decision Point (PDP).

Для того, чтобы вычислять политики безопасности, их нужно где-то создать. За это отвечает другой компонент системы — механизм администрирования политик, в стандарте названный Policy Administration Point (PAP).

Еще одним источником данных, необходимых PDP для вычислений, является источник значений атрибутов, называемый Policy Information Point (PIP).

Последним компонентом является механизм, отвечающий за вызов PDP и правильную обработку его ответа, который именуется Policy Enforcement Point (PEP).

III. АРХИТЕКТУРА МОДУЛЯ УПРАВЛЕНИЯ ДОСТУПОМ ДЛЯ СРЕДЫ NODE.JS

A. Функциональные и технические требования

Необходимо разработать модуль, который включает в себя базовые инструменты для настройки ABAC и CORS в Node.js приложениях. Модуль должен предоставлять веб-интерфейс, через который администраторы сервиса смогут в режиме реального мнения изменять набор правил для различных точек доступа (Endpoints) приложения.

Исходя из общих требований модуль должен удовлетворять следующим функциональным требованиям и требованиям к реализации.

Функциональные требования к модулю:

- Веб-интерфейс для настройки правил доступа;
- Декларативная настройка кросс-доменных запросов для различных источников (CORS);
- Инструменты для реализации авторизации и аутентификации;
- Настройка доступа на основе атрибутов к приложению глобально и для отдельных точек доступа.

Требования к реализации модуля:

- Среда выполнения Node.js;
- Сохранение правил при отключении или перезапуске сервера
- Адаптеры для различных хранилищ (база данных, файл, кэш)
- Веб-интерфейс в виде Single page application
- Адаптеры для подключения к популярным Node.js фреймворках (express, nest, fastify и др.).

Это позволит гибко и своевременно ослаблять или усиливать защиту приложения без изменений в коде и релизов приложения. На рисунке 10 представлена схема архитектуры разрабатываемого модуля

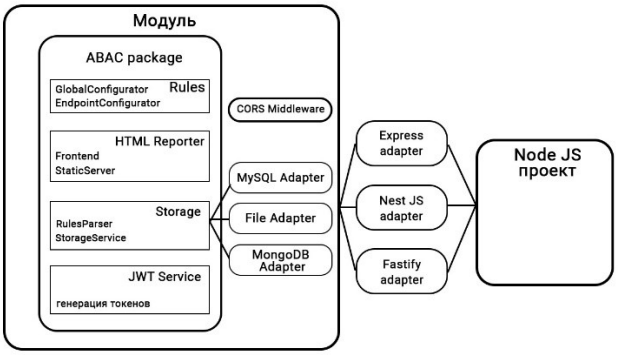

Рис. 10. Архитектура модуля

B. Пакет для конфигурации ABAC

ABAC package – пакет, включающий в себя все необходимые инструменты для настроек правил доступа по атрибутам. Состоит из следующих подмодулей:

1. RulesConfigurator – содержит в себе набор инструментов для формирования правил как для всего приложения, так и для отдельных точек доступа:

- Global configurator – middleware, добавляемый в корень приложений. Через него проходят все запросы и проверяются на соответствие глобальным правилам. Примеры правила:

- Запретить доступ к сервису неавторизованным пользователям.

- Запретить доступ к сервису пользователям, которые не являются работниками филиала компании X и не занимают должность Y.

2. HTML Reporter – Для того, чтобы администрация сервиса могла гибко и своевременно настраивать правила доступа без контакта с программным кодом требуется веб-интерфейс. Пакет содержит в себе статические html, js, css файлы со всем необходимым кодом.

Интерфейс содержит в себе список всех точек доступа к API, например:

- GET /users/:id – точка доступа для получения пользователя по идентификатору
- POST /contracts – точка доступа для создания договора и сохранения его в базу данных.

Для каждой точки доступа можно задать правила, которые после выполнения HTTP запроса сохраняются в хранилище при помощи пакета модуля Storage, о котором будет рассказано далее.

HTML reporter имеет свой конфигуратор, который можно гибко настроить для взаимодействия с бэкендом приложения. Также имеет встроенный static server, который будет раздавать статические файлы для пользователей.

Листинг 2. Пример подключения статического сервера и конфигуратора

```
const HTMLReporter = require('module/frontend')

// Подключение статического сервера, аргументом передается путь, по которому сервер будет раздавать статические файлы
app.use(

HTMLReporter.staticServer('/configuration/rules')
)

app.use(
    HTMLReporter.configuration({
          // Указывается список ролей, для которых доступно изменение правила доступа на основе атрибутов
          availableRoles: ['ADMIN', 'SUPERUSER']
    })
)
```

3. StorageParser – ядро с основными функциями парсинга, обработки, валидации и сохранения правил, которые задают администраторы сервиса через Web интерфейс, описанный ранее.

Ядро хранилища обладает интерфейсом, благодаря которому разработчик сервера может написать адаптер, позволяющий сохранять файл в любое хранилище, будь то база данных, файл или кеш. По умолчанию все правила преобразовываются в JSON формат, который можно сохранить в любое хранилище.

4. JWT service – сервис, который умеет генерировать access и refresh JWT токены со всей необходимой информацией в payload, которая позволяет идентифицировать пользователя и его сессию.

Access токен – короткоживущий токен, который отправляется с каждым запросом и хранит в себе идентификатор пользователя, который невозможно подделать. Позволяет однозначно определить участника сессии. Обычно токен живет 15-30 минут. В случае похищения токена злоумышленник сможет воспользоваться им в течение этого времени, после чего жизнь токена истечет, и он станет не валиден. Токен отправляется в заголовке HTTP запроса Authorization.

Refresh токен – долгоживущий токен, который необходим для перегенерации пары access и refresh токена. Токен живет от 15 до 60 дней. Отправляется в httpOnly secure cookie, которую невозможно изменить или получить через javascript в браузере. Поэтому риск потерять токен сводится практически к нулю.

В сервисе задается секретный ключ, по которому валидируется токен. Ключ должен храниться в защищенном месте и не должен быть потерян, иначе любой токен можно подделать.

Листинг 3. Пример работы с JWT сервисом

```
const JwtService = require('module/jwt)

// Создаем инстанс сервиса
const jwtService = new JwtService({
      secretKey: 'SECRET_PRIVATE_KEY'
```

```
})

// Генерируем пару токенов
const { access, refresh } = jwtService.generate({
     lifetime: { access: '15m', refresh: '30d' },
    payload:  {  userId:  '1',  createDate:
Date.now() }
})

try {
    // Пытаемся декодировать токен, если
целосность была нарушена сервис пробросит
исключение
    const payload = jwtService.decode(token)
} catch (e) {
     // обработка случая с невалидным токеном
}
```

C. *Пакет для конфигурации CORS*

Пакет для работы с CORS представляется из себя конфигурируемый декларативно middleware, поскольку в Node.js приложениях популярен подход с использованием паттерна chain of responsibility (цепочка обязанностей), в частности это реализовано в виде использования middleware, было принято решение разработать пакет для настройки кросс-доменных правил (CORS) в виде промежуточного, конфигурируемого звена. Для всего приложения или для отдельных точек доступа задается whitelist источников, к которым разрешен доступ, а также методы http-запросов, заголовки и прочие CORS настройки.

Интерфейс middleware должен быть максимально простым, а настройка декларативной. Пример представлен в листинге 4.

Листинг 4. Пример глобальной декларативной настройки CORS

```
const corsMiddleware = require('module/cors')

app.use(corsMiddleware([
    {
       origin: 'http://domain1.ru',
       methods: ['GET', 'POST'],
       headers: '*',
    },
{
       origin: 'https://domain2.com',
       methods: ['GET', 'POST', 'PUT', 'DELETE'],
       headers: ['authorization', 'pagination'],
    },
]))
```

D. *Вспомогательные адаптеры*

Помимо этого, разработчики Node.js используют различные фреймворки, которые упрощают и ускоряют разработку. Ядро модуля отвязано от конкретного фреймворка и предназначено для нативного Node.js. В качестве внешних пакетов по схеме архитектуры спроектированы адаптеры, которые содержат в себе базовый функционал для подключения к конкретным фреймворкам. Это позволит упростить подключение модуля, скроет детали реализации и даст продуктовым разработчикам дополнительное временя для разработки под предметную область.

Примеры адаптеров:

1. ExpressAdapter – адаптер для самого популярного фреймворка express для Node.js.
2. NestAdapter – адаптер для enterprise фреймворка для Node.js.

E. *Описание работы модуля в соответствии с XACML*

Веб интерфейс, описанный в архитектуре модуля, представляет из себя Policy Administration Point. Администратор сервиса определяет структуру политик, правил, управляет атрибутами через удобный веб-интерфейс. Любое правило и политику можно удалить или отредактировать через стандартные элементы формы.

Rules configurator является Policy Information Point и Policy Enforcement Point одновременно. При создании инстанса конфигуратора на вход передается набор строго типизированных моделей, с атрибутами которых и будут работать правила. Логика работы конфигуратора проста, на вход он получает название атрибута, например Document.date, а на выходе отдает все значения, которые может найти в текущем контексте выполнения. Это осуществляется за счет заранее описанных схем моделей сущностей, с которыми взаимодействует сервис.

RulesConfigurator является одновременно и местом начала и местом окончания всех действий за счет специфики работы Middleware в Node.js. Именно здесь определяется, проходит ли запрос дальше или блокируется.

Один из самых важных элементов системы является StorageParser. В соответствии с XACML он представляет из себя PDP и именно он отвечает за контроль доступа. Здесь происходит вычисление политик, а также проводятся операции над атрибутами, которые приходят из конфигуратора правил. Помимо этого, StorageParser отвечает за сохранение правил в хранилище.

IV. Заключение

Исходя из вышеизложенного, можно сделать вывод о том, что авторизация и контроль доступа являются одними из самых важных частей современных автоматизированных систем, поскольку напрямую влияют на безопасность и контроль доступа к тем или иным частям системы для разных групп пользователей.

В данной статье были рассмотрены и проанализированы два основных подхода для управления доступом, каждый из которых обладает рядом достоинств и недостатков, стандарт CORS для обмена данными между различными источникам, спецификация XACML для описания правил в ABAC. Также была спроектирована архитектура модуля для платформы Node.js. Модуль содержит в себе все

необходимые инструменты и инфраструктуру, необходимую для решения задач аутентификации, авторизации и контроля доступа на основе атрибутов ABAC.

Библиография

[1] AUTHENTICATION AND AUTHORIZATION OF MOBILE CLIENTS IN PUBLIC DATA NETWORKS/VENKY K., KAN Z., 2002;

[2] David F. Ferraiolo Role-Based Access Control/ D. Richard Kuhn, Ramaswamy Chandramouli, 1992;

[3] Application of Attribute Based Access Control Model for Industrial Control Systems, 2017;

[4] Das S. POLICY ENGINEERING IN RBAC AND ABAC/Sural S., Mitra B., 2018;

[5] Role-Based ABAC Model for Implementing Least Privileges/Zhiguang Q., Javed A., 2018;

[6] Guide to Attribute Based Access Control (ABAC) Definition and Considerations/Vincent C. Hu, Ferraiolo D., 2014;

[7] Proposed NIST Standard for Role-Based Access Control/Sandhu R.., Ferraiolo D., 2001;

[8] XACML Profile for Role Based Access Control (RBAC)/Anderson A.. 2001;

[9] eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01, 2017;

[10] ATTRIBUTE-BASED ACCESS CONTROL MODELS AND IMPLEMENTATION IN

[11] CLOUD INFRASTRUCTURE AS A SERVICE/ Ravi Sandhu, Ph.D., Co-Chair, 2014;

[12] Спецификация Cross Origin Resource Sharing - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS [Электронный ресурс];

[13] L. Huang Protecting Browsers from Cross-Origin CSS Attacks/C. Evans, Z. Weinberg.

Attribute based access control module for cross-origin web requests

T.V. Ulbi, O.R. Laponina

Annotation **– The article analyzes the possibilities of the RBAC and ABAC standards for managing access to resources from various sources. One of the advantages of ABAC is that it allows you to describe an infinite number of different scenarios without the need to create new roles, which allows you to create and modify rules more flexibly. The article highlights the following advantages of ABAC: the ability to describe access control in terms close to the terms of the business logic of the application, the ability to significantly automate the creation of both simple and complex access control rules, including rules with dynamic parameters. The XACML language is used to describe access control rules. Currently, the CORS standard is used to control access to web resources from different sources.**

This article explores the module architecture for the Node.js framework for attribute-based access control for cross-domain origins. The module includes basic tools for setting up ABAC and CORS in Node.js applications. The module provides a web interface with which the service administrator can change the set of rules for various access points (Endpoints) of the application in real-time mode. The module satisfies the following functional requirements: web interface for setting up access rules; declarative configuration of cross-domain requests for different origins (CORS); configuring attribute-based access to both the entire application and individual access points. The description of the module operation in accordance with XACML is given.

Keywords **– RBAC, ABAC, CORS, XACML, PDP, PAP, PEP, Access Control, Node js.**

REFERENCES

[1] AUTHENTICATION AND AUTHORIZATION OF MOBILE CLIENTS IN PUBLIC DATA NETWORKS/VENKY K., KAN Z., 2002;

[2] David F. Ferraiolo Role-Based Access Control/ D. Richard Kuhn, Ramaswamy Chandramouli, 1992;

[3] Application of Attribute Based Access Control Model for Industrial Control Systems, 2017;

[4] Das S. POLICY ENGINEERING IN RBAC AND ABAC/Sural S., Mitra B., 2018;

[5] Role-Based ABAC Model for Implementing Least Privileges/Zhiguang Q., Javed A., 2018;

[6] Guide to Attribute Based Access Control (ABAC) Definition and Considerations/Vincent C. Hu, Ferraiolo D., 2014;

[7] Proposed NIST Standard for Role-Based Access Control/Sandhu R.., Ferraiolo D., 2001;

[8] XACML Profile for Role Based Access Control (RBAC)/Anderson A.. 2001;

[9] eXtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01, 2017;

[10] ATTRIBUTE-BASED ACCESS CONTROL MODELS AND IMPLEMENTATION IN

[11] CLOUD INFRASTRUCTURE AS A SERVICE/ Ravi Sandhu, Ph.D., Co-Chair, 2014;

[12] Specification Cross Origin Resource Sharing - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[13] L. Huang Protecting Browsers from Cross-Origin CSS Attacks/C. Evans, Z. Weinberg.