AUTHENTICATION AND AUTHORIZATION

Ivanov V.[1], Lubova E.[2], Cherkasov D.[3]

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ

Иванов В. В.[1], Лубова Е. С.[2], Черкасов Д. Ю.[3]

[1]Иванов Вадим Вадимович / Ivanov Vadim – студент;
[2]Лубова Елена Сергеевна / Lubova Elena – студент;
[3]Черкасов Денис Юрьевич / Cherkasov Denis – студент,
кафедра компьютерной и информационной безопасности,
Институт кибернетики
Московский институт радиотехники, электроники и автоматики
Федеральное государственное бюджетное образовательное учреждение высшего образования
Московский технологический университет, г. Москва

Аннотация: *аутентификация - это проверка соответствия субъекта и того, за кого он пытается себя выдать, с помощью некой уникальной информации (отпечатки пальцев, цвет радужки, голос и т.д.), в простейшем случае - с помощью имени входа и пароля. Авторизация - это проверка и определение полномочий на выполнение некоторых действий (например, чтение файла /var/mail/eltsin) в соответствии с ранее выполненной аутентификацией. Эти понятия представляют собой своеобразную первую линию обороны, обеспечивающую безопасность информационного пространства организации.*

Abstract: *authentication - is a conformity checking of the subject and the person for whom he is trying to give himself, with the help of some unique information (fingerprints, iris color, voice and so on.), In the simplest case - via a login and password. Authorization - this is a test and determination of the powers to perform certain actions (such as reading the file /var/mail/eltsin) in accordance with the previously performed authentication. These concepts represent a kind of first line of defense, ensuring the safety of information space organization.*

Ключевые слова: *аутентификация, авторизация, токен, сервер.*
Keywords: *authentication, authorization, token, server.*

Аутентификация является, пожалуй, наиболее распространенным требованием любого приложения. Возможность быстро и легко зарегистрироваться или войти на сайт играет огромную роль для пользователя. В традиционном веб-приложении это обычно делается с помощью сервера, который отслеживает сеансы в той или иной форме. Статические приложения также могут отслеживать сессии, но работа клиента в непроверенной среде означает, что должны быть приняты дополнительные меры предосторожности, чтобы гарантировать, что пользователь проверен и заслуживает доверия.

Прежде чем мы продолжим, важно прояснить различие между аутентификацией и авторизацией. Аутентификация представляет собой процесс проверки того, что пользователь является тем, кем есть; это типичная регистрация в системе. Авторизация проверяет, что пользователи могут выполнять определенные действия, которые вы указали, и не более. Нужно убедиться, что пользователи не могут читать личные данные других лиц и выполнять административные действия.

Что такое непроверенная среда

Одним из самых больших различий между клиентской и серверной частями кода веб-приложения является то, что конечный пользователь может запустить произвольный код на стороне клиента без предварительного разрешения. Это означает, что все действия, которые может сделать ваше приложение с помощью JavaScript, пользователь может легко воспроизвести, открыв Инструменты разработчика в своем браузере. Это влечет за собой две проблемы:

- Авторизация в приложениях не должна размещаться внутри кода на стороне клиента, так как он может быть произвольно изменен любой стороной. (Либо авторизация в приложениях должна размещаться без возможности внесения правок со стороны клиента, поскольку иначе он может быть произвольно изменен любой стороной.)

- Код приложения на стороне клиента не может содержать «секретную» информацию, такую как ключи API или любого рода учетные данные для доступа, которые не являются специфическими для текущего пользователя.

Поскольку ненадежные среды создают проблемы и обнаруживают уязвимые места, стоит сосредоточиться на создании безопасной архитектуры. Статические приложения не могут позволить себе отказаться от обеспечения какой-либо защиты. Так что же делать?

Одно из лучших правил заключается в том, чтобы предоставить пользователям достаточную свободу действий, чтобы они могли уничтожить свои собственные данные, но не чужие. Очень важно помнить о том, что вы защищаете свое приложение от злоумышленников, а не пытаетесь необоснованно ограничить пользователей. Нет ничего страшного в том, что пользователь хочет взломать приложение и в процессе уничтожает свои собственные данные учетной записи пользователя. Нужно беспокоиться только тогда, когда его действия могут повлиять, скомпрометировать или уничтожить данные других пользователей.

Процесс аутентификации

Если вы не строите автономное приложение, предназначенное для хранения данных только в локальном браузере, ваш процесс аутентификации должен включать сервер. Самое простое средство аутентификации работает в основном так:

1. Клиент направляет пользователя к процессу аутентификации на стороне сервера. Это может быть сделано через всплывающее окно JavaScript или перенаправление браузера включается установкой window. location.
2. Сервер проверяет подлинность пользователя с помощью пароля, цифровых подписей или других средств.
3. Сервер создает случайно сгенерированный токен (ключ) и связывает его с уже прошедшим аутентификацию.
4. Сервер передает токен обратно клиенту.
5. Клиент использует предоставленный токен при последующих запросах к серверу в качестве документа, удостоверяющего личность, предоставляющего пользователю доступ к защищенным ресурсам.

Поскольку токен генерируется в момент входа в систему случайным образом, его присутствие служит достаточным доказательством того, что запрос исходит от пользователя, которому был выслан токен. Токен, который предоставляет доступ без каких-либо дополнительных требований, известен как токен носителя [1].

До того, как идентификация личности через токены стала распространенной, многие API-интерфейсы приложений использовали только имя пользователя и пароль учетных данных, передаваемых по запросу. Проверка подлинности через токен гораздо лучше подходит для такой системы по ряду причин:

1. Полномочия, особенно предоставленные пользователем пароли, должны храниться так надежно, как это возможно. Особенно, если приложение предоставляет доступ третьим лицам к серверу ресурсов, прямое использование пароля приведет к падению системы безопасности, что может поставить под угрозу аккаунты пользователей за пределами вашего приложения.
2. Сервер может отменить доступ к сгенерированному токену в любое время. Если у пользователя есть основания полагать, что один из авторизированных токенов был в некотором роде скомпрометирован, можно легко отменить доступ к токену без универсального сброса учетных данных пользователя или все предоставленных токенов.
3. Помимо ручного аннулирования доступа, у токенов может автоматически закончиться срок действия, что может потребовать дополнительных доказательств того, что они должны оставаться в силе. По сути, проверка подлинности на основе токенов дает вам возможность полностью контролировать аутентификацию и авторизацию приложения.

Обратите внимание на то, что каждый, кто предъявляет токен, немедленно получает доступ к данным. Поэтому жизненно важно, чтобы любое сообщение, которое включает токен, проходило через соединение, защищенное SSL-шифрованием.

Аутентификация и авторизация требуют обширных знаний, а потому вам нужно будет серьезно задуматься над стратегией, которую вы решите использовать. Простое предъявление токена кажется наилучшим выходом из ситуации, но эта система слишком уязвима. Если она будет перехвачена, то сможет полностью скомпрометировать учетную запись пользователя. Вот несколько советов, которые можно учесть для того, чтобы создать более высокий уровень безопасности пользователя для статического приложения [2]:

1. Не изобретайте колесо. Люди потратили много времени, думая на эти темы. Лучше всего придерживаться существующего протокола OAuth 2.0, а не пытаться придумать свой собственный.
2. Не позволяйте доступу к вашим токенам длиться вечно. Обновляйте токены или иным образом периодически требуйте повторную проверку подлинности, чтобы гарантировать, что устаревшие учетные данные клиента не станут основанием для нарушения безопасности.

3. Не храните токены доступа в незашифрованном виде или на локальном диске.

4. Используйте области авторизации. Когда клиент выполняет проверку подлинности, запросите список конкретных ресурсов, к которым он хочет получить доступ, даже если он полностью под вашим контролем.

Трехсторонняя аутентификация

В зависимости от потребностей вашего приложения, вы можете не реализовать свою собственную проверку подлинности пользовательских данных на основе токена. Если вы создаете приложение, которое в первую очередь взаимодействует с другой службой, могут быть доступны библиотеки JavaScript (например, Facebook's JS SDK) или же они могут предложить способы создания авторизации токенов вручную (например, GitHub's API).

Также появился относительно новый вариант, популярность которого растет, - он является фоновым провайдером аутентификации. Такие сервисы, как Firebase и UserApp, предлагают простые способы аутентификации пользователей статического приложения без генерирования кода на стороне сервера. Каждый сервис будет иметь свои сильные и слабые стороны, которые должны рассматриваться как часть общей стратегии аутентификации.

Авторизация пользователя (предоставление ему прав доступа к ресурсам) является положительным результатом аутентификации помимо установления доверительных отношений и выработки сессионного ключа. Процедура аутентификации необходима для обмена информацией между компьютерами. Для обеспечения защиты линии связи от прослушивания или подмены одного из участников взаимодействия используются сложные криптографические протоколы.

Литература

1. Под редакцией А. А. Шелупанова, С. Л. Груздева, Ю. С. Нахаева. Аутентификация. Теория и практика обеспечения доступа к информационным ресурсам. М.: Горячая линия – Телеком, 2009. С. 552.
2. *Смит Ричард Э.* Аутентификация: от паролей до открытых ключей. М.: Вильямс, 2002. С. 432.

CHECKRECIPIENT SOLUTION OVERVIEW FOR EMAIL SECURITY

Patrakov E.[1], Patrakova D.[2]

ОБЗОР РЕШЕНИЯ CHECKRECIPIENT ДЛЯ БЕЗОПАСНОСТИ ЭЛЕКТРОННОЙ ПОЧТЫ

Патраков Е. И.[1], Патракова Д. И.[2]

[1]Патраков Евгений Иванович / Patrakov Evgeny – студент,
департамент менеджмента;
[2]Патракова Дарья Ивановна / Patrakova Daria – студент,
кафедра теоретических основ радиотехники,
Уральский федеральный университет им. Б. Н. Ельцина, г. Екатеринбург

Аннотация: *неверно адресованное письмо в современном мире вследствие непреднамеренной человеческой ошибки может стать результатом утечки конфиденциальных данных. Для решения данной проблемы были предложены следующие механизмы: «задержка отправки сообщения», «шифрование письма». Однако такие методы имеют существенные недостатки: уменьшение скорости отклика электронной почты, сложность реализации. Вследствие этого было предложено решение CheckRecipient, которое имеет две реализации: CheckRecipient Guardian, обнаруживающий и предотвращающий угрозы по электронной почте, используя искусственный интеллект, и CheckRecipient RuleBuilder, обнаруживающий и предотвращающий угрозы по электронной почте, используя машинное обучение.*

Abstract: *incorrectly addressed letter in the modern world can become result of leakage of confidential data. To solve this problem, the following mechanisms have been offered: «delay of sending message», «encryption of a letter». However, such methods have significant drawbacks: reduction in the response speed of the email, the complexity of the implementation. As a result, it was suggested solution CheckRecipient, which has two implementations: CheckRecipient Guardian, detect and prevent threats by e-mail, using artificial intelligence, and CheckRecipient RuleBuilder, detect and prevent threats by email, using machine learning.*