УДК 004.056.5

Стахов Р.О.
Национальный технический университет Украины
«Киевский политехнический институт имени Игоря Сикорского»

АНАЛИЗ УЯЗВИМОСТЕЙ В АЛГОРИТМАХ ЦИФРОВОЙ ПОДПИСИ ТЕХНОЛОГИИ JSON WEB TOKEN

В статье приводятся существующие методы цифровой подписи данных технологии Json Web Token, анализируются механизмы и алгоритмы их работы. Описываются уязвимости существующих методов цифровой подписи, а также способы защиты и нивелирования возможного вреда, причиняемого данными уязвимостями. Приводятся примеры наиболее часто используемых злоумышленниками способов взлома и завладения конфиденциальной информацией с использованием нижеуказанных недочетов.

Ключевые слова: *уязвимость, защита, алгоритм, цифровая подпись, взлом, библиотеки.*

Постановка проблемы. Стандарт JSON, реализующий механизм аутентификации и авторизации при помощи веб-токенов, имеет большое количество реализаций, среди которых есть библиотеки, имеющие критические уязвимости, позволяющие злоумышленникам обойти систему аутентификации. К примеру, использование библиотек node-jsonwebtoken, pyjwt, namshi/jose, php-jwt или jsjwt с использованием асимметричных ключей (RS256, RS384, RS512, ES256, ES384, ES512) имеют уязвимости, которые в достаточной степени нивелированы лишь в самых последних версиях. Недостаток системы аутентификации был обнаружен в нескольких реализациях библиотек JWT на разных языках программирования. В данной статье будет указано, где именно возникают проблемы и каким образом их можно предотвратить.

Анализ последних исследований и публикаций. Технология цифровой подписи токенов доступа в JWT является открытой, все алгоритмы ее работы находятся в свободном доступе. Компаниями seedbox, 0Auth и Adobe были выявлены несколько уязвимостей в библиотеках, написанных для внедрения технологии JWT на различных языках программирования, включая сетевые и десктопные приложения. В частности, это возможность несанкционированного обхода процесса аутентификации пользователя при создании токена JWT, а также механизмы взлома цифровой подписи самого токена JWT.

Постановка задачи. Для получения несанкционированного доступа к конфиденциальной информации, защищенной технологией JWT, злоумышленники применяют 2 подхода: использование уязвимостей реализации технологии в различных библиотеках и фреймворках, а также использования слабых мест в самих алгоритмах цифровой подписи с использованием стороннего программного обеспечения.

В статье описываются варианты подмены злоумышленником информации об алгоритме цифровой подписи конкретного токена с использованием уязвимостей конкретных библиотек, а также взлом зашифрованных токенов с использованием стороннего программного обеспечения.

Примеры уязвимостей в реализациях технологии JWT:

1) библиотека namshi/jose для использования в PHP имеет серьезную уязвимость в генераторах асимметричных ключей алгоритмов RS256 – ES512;

2) библиотека node-jsonwebtoken для JavaScript не имеет встроенного механизма контроля выбора алгоритма шифрования информации в процессе создания токена;

3) на многих серверах, в зависимости от конкретной библиотеки JWT и загруженности самого сервера, либо не всегда контролируется длина ассиметричного ключа, необходимая для надежного шифрования конфиденциальной информации, либо же не проверяется корректный выбор алгоритмов шифрования цифровой подписи.

Как известно, JSON Web Token (JWT) является стандартом для создания токенов, которые

используются в качестве средства аутентификации. К примеру, сервер может генерировать токен, который имеет запрос «зарегистрирован как администратор» и предоставляет эти возможности клиенту. Затем клиент может использовать этот токен при авторизации, чтобы доказать, что он вошел в систему как администратор. Токены подписываются ключом сервера, поэтому сервер может проверить, его валидность. Но если сам алгоритм цифровой подписи токена явно не указан и это не было проверено на стороне сервера, злоумышленник имеет возможность подменить алгоритм конкретного токена выгодным ему, тем самым получив доступ к данным авторизации, находящимся в токене, что создаст дыру в системе серверной безопасности.

Изложение основного материала исследования. Сначала будет продемонстрирована возможность сторонней подмены алгоритма цифровой подписи. Как известно, JWT состоит из трех частей: заголовка, полезной нагрузки и цифровой подписи. Часть заголовка определяет, какой алгоритм используется для генерации цифровой подписи, и выглядит примерно так:

header = '{"alg":"HS256","typ":"JWT"}';

HS256 указывает, что этот токен подписывается с использованием алгоритма шифрования HMAC-SHA256.

Полезная нагрузка содержит данные для доступа:

payload = '{"loggedInAs":"admin",
"iat":1422779638}'

Как было указано в спецификации JWT, используется временная метка iat, что значит «issued at». Подпись вычисляется с помощью алгоритма base64url, кодирующего заголовок и полезную нагрузку, и объединяющего их с символом точки в качестве разделителя:

key = 'secretkey'
unsignedToken = encodeBase64(header) + '.
' + encodeBase64(payload)
signature = HMAC-SHA256(key, unsignedToken)

В конечном итоге формируется токен:

token = encodeBase64(header) + '.
' + encodeBase64(payload) + '.
' + encodeBase64(signature)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYX
QiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN_
oWnFSRgCzcmJmMjLiuyu5CSpyHI

Уязвимость состоит в том, что в поле «alg» можно вставить значение «none»:

header = '{"alg":"none","typ":"JWT"}';

Алгоритм предназначен для использования в ситуациях, когда валидность токена уже была проверена. Алгоритм HMAC-SHA256 – один из двух алгоритмов, которые обязательны для реализации во всех библиотеках (другой – HS256).

К сожалению, некоторые библиотеки проводят обработку токенов, подписанных со значением алгоритма «none» как валидных токенов с проверенной подписью, прошедших валидацию (pyjwt, namshi/jose). В результате данной уязвимости любой может создавать свои собственные «подписанные» токены с любой полезной нагрузкой, которые они хотят, позволяя получить произвольный доступ к учетной записи в некоторых системах.

Создание такого токена весьма простое. Необходимо изменить приведенный выше заголовок так, чтобы он содержал «alg»: «none» вместо «HS256».

Несколько библиотек, указанных выше, до сих пор имеют дыры в этой области. Они пошли другим путем: если в токене был предоставлен секретный ключ, то его проверка не будет выполняться с использованием алгоритма none. Это хорошая идея, но она не решает основной проблемы: злоумышленники контролируют выбор алгоритма.

Спецификация JWT также определяет ряд асимметричных алгоритмов подписи (на основе RSA и ECDSA). С помощью этих алгоритмов токены создаются и подписываются с использованием закрытого ключа, но проверяются с использованием соответствующего открытого ключа. Это довольно аккуратно: если пользователь публикует открытый ключ, но сохраняет личный ключ для себя, только он может подписывать токены, но каждый может проверить, правильно ли сам токен был подписан.

Большинство библиотек JWT имеют данный метод:

verify(string token, string verificationKey)

В системах, использующих подписи HMAC, проверка ключа будет происходить с помощью секретного ключа подписи сервера (поскольку HMAC использует тот же ключ для подписи и проверки):

verify (clientToken, serverHMACSecretKey)

В системах, использующих асимметричный алгоритм, для проверки ключа будет использоваться открытый ключ, с которым должен быть проверен токен:

verify (clientToken, serverRSAPublicKey)

К сожалению, злоумышленник может воспользоваться этим. Дело в том, что не происходит

никакой проверки на соответствие алгоритма подписи и анализа подлинности. Иными словами, нет контроля информации насчет того, какой именно алгоритм был использован для подписи. Если сервер ожидает токен, подписанный с помощью алгоритма RSA, но фактически получает токен, подписанный алгоритмом HMAC, он будет считать, что открытый ключ на самом деле является секретным ключом HMAC.

Это весьма серьезная уязвимость. Секретные ключи HMAC должны быть закрытыми, в то время как открытые ключи – общедоступными. Это означает, что злоумышленник имеет доступ к открытому ключу и может использовать его для создания токена, который будет принимать сервер.

Это осуществляется довольно просто. В любой библиотеке JWT выбирается полезная нагрузка для токена. Затем создается открытый ключ, используемый на сервере в качестве ключа проверки (скорее всего, в текстовом формате PEM). Наконец, подписывается сам токен, используя открытый ключ PEM-формата в качестве ключа HMAC. По существу:

forgedToken = sign(tokenPayload, 'HS256', serverRSAPublicKey)

Самая сложная часть – убедиться, что параметр serverRSAPublicKey идентичен ключу проверки, используемому на сервере. Строки должны точно соответствовать атаке, чтобы работать – точно такой же формат, и никаких лишних или отсутствующих разрывов строк.

В конечном итоге любой, кто знает открытый ключ, может подделать токены доступа, которые легко пройдут проверку на сервере, тем самым дав злоумышленнику несанкционированный доступ к защищенным данным.

Таким образом, для того, чтобы избежать возможного взлома токена, необходимо предпринять следующие шаги: напрямую проверять выбор алгоритма цифровой подписи и генерацию секретного ключа для токена, если вы разрабатываете программы с использованием JWT, а также использовать (к примеру, для облачного хранения данных) лишь те сервера, которые имеют новейшие версии библиотек JWT.

Второй способ взлома цифровой подписи токена состоит в использовании стороннего программного обеспечения методом подбора секретного ключа. При недостаточной надежности ключи шифрования легко поддаются атаке полным перебором (bruteforce).

При создании цифровой подписи используются 3 наиболее распространеных алгоритма:

HMAC + SHA256 (HS256)

RSASSA-PKCS1-v1_5 + SHA256 (RS256)

ECDSA + P-256 + SHA256 (ES256)

Код аутентификации сообщения на основе хеша (HMAC) – это алгоритм, который объединяет определенную полезную нагрузку с секретом с использованием криптографической хэш-функции, такой как SHA-256. Результатом является код, который может использоваться для проверки сообщения только в том случае, если обе генерирующие и проверяющие стороны знают секрет. Другими словами, HMAC позволяют проверять сообщения через общие секреты.

Ниже указан пример демонстрации алгоритма подписи на основе HMAC:

```
const encodedHeader = base64(utf8(JSON.stringify(header)));
const encodedPayload = base64(utf8(JSON.stringify(payload)));
const signature = base64(hmac(`${encodedHeader}.${encodedPayload}`, secret, sha256));
const jwt = `${encodedHeader}.${encodedPayload}.${signature}`;
var jwt = require('jsonwebtoken');
const payload = {
sub: "1234567890",
name: "John Doe",
manager: true};
const secretKey = 'secret';
const token = jwt.sign(payload, secretKey, {
algorithm: 'HS256',
expiresIn: '10m' });
```

Уже существует множество утилит и программ на многих языках, которые используются для скоростного подбора секретных ключей токенов. Они свободно распространяются на сетевых репозиториях, например, GitHub. Обладая огромной вычислительной мощью компьютера, эти инструменты могут без проблем подобрать секретный ключ HS256 JSON.

Следует обратить внимание, что в стандарте защиты RFC7518 указано, что «только ключ с таким же размером, как и хэш-вывод (например, 256 бит для HS256) или больше, должен использоваться с этим алгоритмом».

При использовании утилит типа jwt-crack, генерирующих секретный ключ при помощи «грубой силы», возможен взлом токена, если длина ключа недостаточна.

К примеру, токен вида:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOn

RydWV9.cAOIAifu3fykvhkHpbuhbvtH807-Z2rI1FS3vX1XMjE

С секретным ключом из 8 бит «Sn1f» при использовании указанной выше утилиты способен произвести подбор ключа менее, чем за 1 минуту.

При использовании ключа большей, но недостаточной длины (к примеру, слово «secret») время подбора увеличивается до 60-76 минут, в зависимости от конкретной утилиты подбора ключей.

Давайте еще раз взглянем на ключи, которые мы использовали для создания токенов, которые были легко взломаны. Каковы ключевые размеры? Первый ключ «Sn1f» – 32-разрядный. (1 символ = 8 бит)

Второй ключ, «secret» имеет длину 48-бит. Их размеры слишком малы для использования в цифровой подписи.

Поэтому при создании маркера JSON Web следует использовать цифровую подпись именно алгоритмом HS256, чтобы иметь возможность сгенерировать секретный ключ надлежащего размера. Секретные ключи Auth0 имеют длину 512 бит и не подвержены возможности взлома при помощи грубой силы.

Выводы. Технология JSON Web Token (JWT) являются весьма эффективной и может легко использоваться на большинстве современных платформ и языков программирования. Это надежный способ передачи подписанной или зашифрованной информации между приложениями.

Но она не лишена своих недостатков. В статье были приведены варианты обхода системы безопасности 2 путями: подменой алгоритма цифровой подписи данных технологии и взломом самого алгоритма при помощи подбора секретного ключа методом «грубой силы».

Были проанализированы и описаны возможные способы защиты от вышеприведенных атак и конкретные шаги по нивелированию возможного вреда, причиняемого данными уязвимостями.

Список литературы:

1. Johnson M: New advanced personal data protection/ Wiley Information Technologies, 2016.
2. Troelsen I: JWT view via C# libraries. Apress. 2016.
3. Albahari J: JWT – a brand new protection system. Bookjoy. 2015.

АНАЛІЗ ВРАЗЛИВОСТІ В АЛГОРИТМАХ ЦИФРОВОГО ПІДПИСУ ТЕХНОЛОГІЇ JSON WEB TOKEN

У статті наводяться методи цифрового підпису даних технології Json Web Token, аналізуються механізми й алгоритми їх роботи. Описуються уразливості методів цифрового підпису, а також способи захисту й нівелювання можливої шкоди, завданої цими уразниками. Наводяться приклади найбільш часто використовуваних зловмисниками способів зламу і заволодіння конфіденційною інформацією з використанням нижчезазначених недоліків.

Ключові слова: *вразливість, захист, алгоритм, цифровий підпис, злам, бібліотеки.*

VULNERABILITY ANALYSES OF THE JSON WEB TOKEN DIGITAL SIGNATURE ALGORITHM

The article describes the existing methods of digital signature data technology Json Web Token, analyzes the mechanisms and algorithms for their work. The vulnerabilities of existing digital signature methods are described, as well as ways to protect and level out the possible harm caused by these vulnerabilities. Examples of the most frequently used methods by hackers to crack and grab confidential information using the following shortcomings.

Key words: *vulnerability, protection, algorithm, digital signature, cracking, libraries.*