****

**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**Факултет по компютърни системи и технологии  
Компютърно и софтуерно инженерство**

**Реферат**

**Програмни технологии за сигурен код**

**Тема:**

***Уязвимости и атаки за JSON Web Token стандарта***

Изготвил:

Иван Ивайлов Милев

Група : 51

Фак.№ 121218022

Съдържание

[Въведение 3](#_Toc99814147)

[Начин на работа 3](#_Toc99814148)

[Структура 3](#_Toc99814149)

[Заглавна част: 3](#_Toc99814150)

[Полезна информация: 3](#_Toc99814151)

[Подпис 4](#_Toc99814152)

[Алгоритъм на използване 5](#_Toc99814153)

[JSON Web token уязвимости и атаки 6](#_Toc99814154)

[Уязвимости 6](#_Toc99814155)

[Грешка при валидацията на подписа 6](#_Toc99814156)

[Позволен алгоритъм None (без алгоритъм) 6](#_Toc99814157)

[Атаки 7](#_Toc99814158)

[Инжекция чрез kid параметъра 7](#_Toc99814159)

[Атаки използващи jku 10](#_Toc99814160)

[Заключение 11](#_Toc99814161)

[Демонстрация 11](#_Toc99814162)

[Използвана литература 12](#_Toc99814163)

# Въведение

В днешно време интернетът е на практика на всякъде. Вече интернетът е част от почти всяка част от живота, от обикновена комуникация до банкови трансфери. С важността на информацията разбира се е нужна и необходимата сигурност, за да няма злоупотреби от недоброжелатели.

JSON Web Token е инструмент от категорията за удостоверяване – автентикация и управление на правата на потребителите. С негова помощ се защитават данните в сървърите от това някой да ги достъпи с непълни или некоректни права.

# Начин на работа[1]

JSON web token е отворен стандарт [https://tools.ietf.org/html/rfc7519], който дефинира компактен и самостоятелен начин да се прехвърлят данни в JSON формат между две системи.

Тази информация може да бъде проверена, защото е дигитално подписана. Подписа може да бъде направен с „тайна“ (secret) или с двойка публични и частен ключове с алгоритми от вида RSA или ECDSA. Друг вариант за постигане на сигурна комуникация между двете системи използвайки JSON Web token е чрез криптиране. Важно е да се отбележи, че подписаните token-и могат да проверяват за интегритета на данните, докато криптирането крие данните от някой, който може да ги засече.

За случаите, в които се използват двойка ключове, се извършва и допълнителна проверка, за това дали подписа е направен с поверителния ключ, а не с публичния.

## Структура

Структурата на JSON Web Token-а е изключително проста и компактна. Тя се състои от три части:

* Header – Заглавна част
* Payload – полезна информация, тяло
* Signature – подпис

Тези три части са конкатенирани с точка, което значи, че типичният резултат е от вида:

xxxx.yyyy.zzzz

### Заглавна част:

Заглавната част от своя страна се състои от две части. Те са типът на токъна, който е JWT в случая и алгоритъма, който е използван, за да се подпише токъна.

Неговият вид е от типа:

{

“alg”: “HS256”,

“typ”: “JWT”

}

След като е конструиран JSON-а, резултата се кодира с алгоритъма Base64 и вече кодираното съобщение се добавя като първият низ от xxxx.yyy.zzz на мястото на x-овете.

### Полезна информация:

В тази част се съдържа информацията за субекта на токена заедно с допълнителна информация. Информацията за субекта се нарича иск и има три вида искове: регистрирани, публични и частни.

Регистрираните искове са предефинирани, те не са задължителни, но са препоръчителни.

Публичните искове могат да се дефинират от използващите стандарта, но за да се избягват дублирания, трябва да се добавят и в списъка на ИАНА (<https://www.iana.org/assignments/jwt/jwt.xhtml>) или да се дефинират като част от URI с пространство на имената, което да не позволява колизии.

Частните искове са персонализирани искове, които се използват за прехвърлянето на реалните данни, които интересуват двете страни участващи в комуникацията.

Примерен вид на тази част от JSON Web Token-а е следния:

{  
 “sub”: “registered claim”,   
 “name”: “Alice Bob”,   
 “admin”: “true”   
}

Отново както и миналата, част JSON-а се кодира с Base64Url и вече кодираното съобщение се добавя в низа xxxx.yyyy.zzzz на мястото на yyyy.

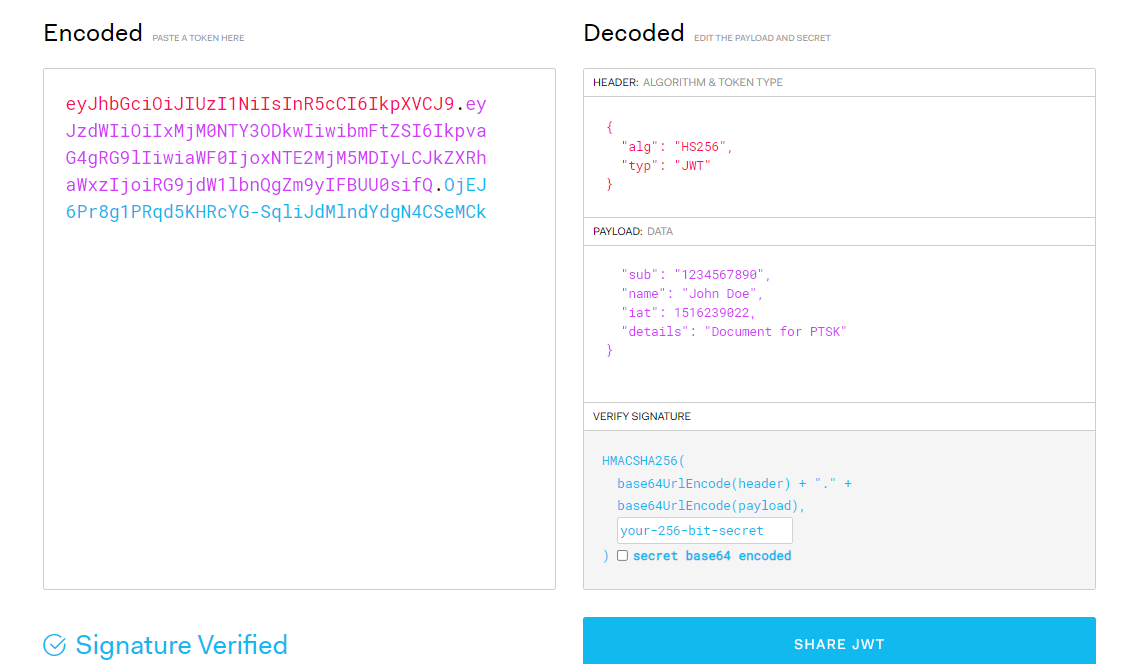
### Подпис

За да се подпише JWT заявката са нужни кодираната заглавна част, кодираната полезна информация, алгоритъмът, който е зададен в заглавната част и тайна. С зададения алгоритъм, ще бъдат създаден подписа. Пример за подпис с алгоритъма HMAC SHA256:

HMACSHA256(encodedHeader + “.” + encodedPayload, secret)

Подписът се използва, за да валидира, че съобщението е в оригиналното си състояние, в което е било изпратено от източника. Ако токенът е подписан с частен ключ то тогава ще се провери и това дали изпращача е правилния или има опит за имперсонация.

След като и трите части от JSON Web Token-а са налични остава само тяхното сглобяване. След като се сглобят, както е описано по-горе, се получава един стринг с две точки, които разделят трите фрагмента на токена, както е показано на фигура 1.



Фигура 1

## Алгоритъм на използване

Както при всяко автентикацияя, потребителя успешно се вписва при въведени правилни креденшъли. В този случай за правилно въведените от потребителя данни се връща JSON Web token, който да бъде използван за по-нататъшна автентикация. Това означава че този вид токън сам по себе си е креденшъл затова трябва да не се пази по-дълго от необходимото и да се пази на сигурно място, а не например в бисквитките на браузъра.

След като потребителят успешно се е вписал и клиента има JSON Web token за всяка последваща заявка към ресурс или път, който изисква някакво ниво на достъп трябва да се подава този токън. Най-често това се прави в заглавната част на заявката в Authorizaion header-а.

Това изглежда по следния начин:

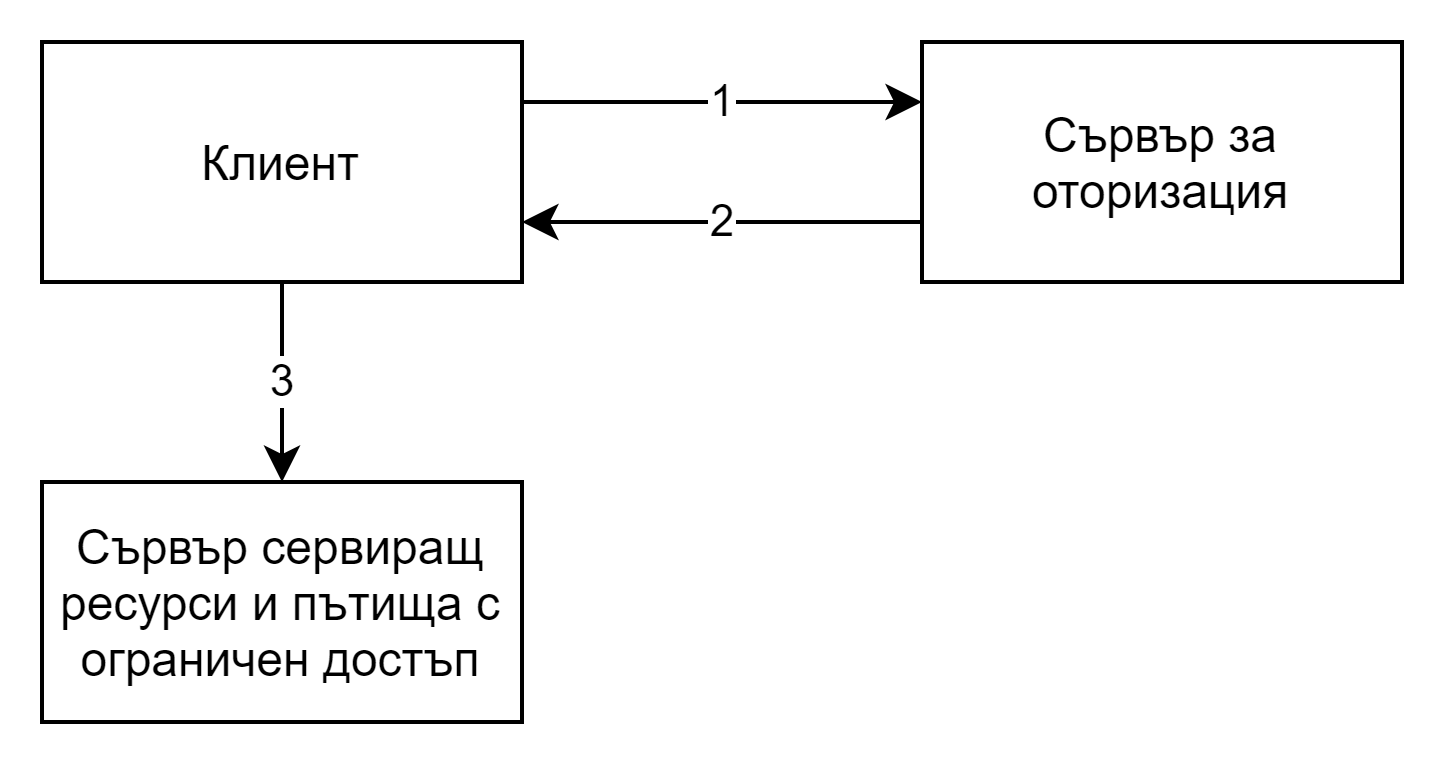
Authorization: Bearer <JSON Web token>

След изпращането на заявката от клиента, сървъра ще валидира токъна съхранен в заглавната част на заявката и ако токъна съществува и токъна има нужните права за достъп сървърът ще обработи заявката.

Ако JSON Web token-а се изпраща посредством http е важно да се отбележи, че много сървъри имат максимален размер за заглавната част на заявките – обикновено 8 KB.

Целият процес на работа се свежда до три стъпки, които могат да се видят на фигура 2:

1. Клиента прави заявка за оторизация с данните на потребителя към специализиран сървър за оторизация.
2. При правилни данни за потребителя сървъра за оторизация връща като отговор JSON Web Token
3. Клиента може да използва получения JSON Web Token, за да достъпва ресурси и пътища, за които се изискват права, които има



Фигура 2

# JSON Web token уязвимости и атаки

## Уязвимости[3]

Както бе разгледано в начина на работа на JSON Web token-ите, те са разработени, така че да бъдат гъвкави и разширяеми, което дава голяма свобода на разработчиците. Тази свобода крие и своите опасности с много възможности за грешки при имплементацията и експлоатацията. Най-често срещаните такива са неправилно валидиране на токена или като цяло изпускане на валидацията. Друга често срещана грешка е позволяването на алгоритъм от тип None, често пъти се бъркат алгоритмите, които се използват за криптиране. Последната доста тривиална грешка е използването на лесна тайна, с която да се подписват токъните. Всяка една от тези уязвимости ще бъде разгледана по-детайлно.

### Грешка при валидацията на подписа

Този вид грешки е един от най-рядко срещаните, защото бързо и лесно се забелязва и отстранява. В много случаи програмистите използват готови библиотеки за логиката обвързана с трансформирането на данните в токън, съответно при получаване на токън се използват методи от вида на decode() и verify(). Много библиотеки викат verify() с извикването на decode(), но в някои случаи това не е вярно и програмистът забравя да верифицира токъна и по този начин може и с произволен подпис да се сдобие с достъп до машината.

### Позволен алгоритъм None (без алгоритъм)

Както вече бе споменато в JSON Web Token-а подписа се прави с конкретен алгоритъм. Има възможност да не се използва такъв алгоритъм. Това на практика означава, че може да се пропусне подписа.

Ако има следния токен:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoidXNlcjEyIiwiaXNfYWRtaW4iOmZhbHNlLCJleHAiOjE2NDg5MDI0MTN9.1WS02WI7E6BVGm9RX7h6acsUb9fvNbPGRi33b7L29eM

И се декодира ще се видят първите две части, само подписа ще остане неясен:

{"typ":"JWT","alg":"HS256"}{"name":"user12","is\_admin":false,"exp":1648902413}db;UoQ\_zio5F-o

При позволен алгоритъм None просто може да се махне последната част от токена (подписа) и да се модифицира останалата част, в случая например is\_admin да стане true, и alg: “None”. След като се направи новото тяло то ще има вида:

eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJuYW1lIjoidXNlcjEyIiwiaXNfYWRtaW4iOnRydWUsImV4cCI6MTY0ODkwMjQxM30=.

Реално този подход ще работи и при грешна проверка на подписа, тъй като при онзи случай на практика се „пропуска“ подписа.

Примерен лош код на python, който позволява всички алгоритми и не валидира подписа:

 try:

    data = **jwt**.decode(token, app.config['SECRET\_KEY'], algorithms=["HS256", "RS256"])

except:

    try:

        data = **jwt**.decode(token, options={"verify\_signature": False})

    except:

        return **jsonify**({'message': 'token is invalid'})

За да се избягват тези уязвимости винаги трябва да се проверява дали са спазени всички препоръки от стандарта и да се проверява както ръчно така и с автоматизирани тестове дали се спазват. Препоръките за това как да се използва стандартът може да се видят в официалната статия за стандарта – https://datatracker.ietf.org/doc/html/rfc7519#section-12.

## Атаки[3][4][5]

### Инжекция чрез kid параметъра

В много случаи kid (Key Id) параметъра се използва, за да се вземе тайната, с която е направен подписа. Това най-често се случва като ключа се взима от база или от файл на файловата система. Тъй като ключа се използва за достъп до данни той е потенциално слабо място за системата, ако тя не е подсигурила проверка за специални символи, коментари и т.н. в него. Този вид атака се цели в това да замени тайната, която се използва при декриптиране с тайна, която е използва за криптиране на нов токен, който има нужните права за достъп до ресурс с ограничен достъп.

Тези атаки винаги започват с декодиране на токена и проверка дали в него има kid. Ако това е вярно, то от стойността на ключа се опитва да се подразбере за какво се използва: ако е число или GUID вероятно идентификатор в базата, който дава тайната за подписа, ако е някакъв вид път е възможно тайната да се съхранява във файл на системата.

Пример за заглавна част на токъна, която би подсказала за вземане на тайната от файл:

{

"alg": "HS256",

"typ": "JWT",

"kid": ./secretKey "

}.

Знаейки това, то много лесно е възможно да се пробва изпълнението на код в терминала като kid стойността се замени с нещо от вида: "key1|/usr/bin/uname"

Друг вариант е ако се използва път до файл, да се навигира до някой системен файл, който е със съдържание, което е ясно за хакера, по този начин е възможно да се направи заявка с тайна, която отговаря на съдържанието на файла и така да се замени подписа. Пример

Kid: “/etc/init.d”

#### SQL инжекция чрез kid

За да се разбере дали такава атака би била осъществима се започва с декодиране на получения от сървъра токен:

{

"token":“eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6IjgifQ.eyJpc19hZG1pbiI6ZmFsc2UsIm5hbWUiOiJ1c2VyMTIifQ.txlNxF07RhuGA9k-1rFG5dnqpzlz041HKeVYuNq2Izk"

}

След декодиране се вижда:

{"typ":"JWT","alg":"HS256","kid":"7"}

{"is\_admin":false,"name":"user12" }

И нищо смислено за подписа:

DRt

,o5k

Ɂ\C

Забелязва се че kid участва и при това е число, вероятно индекс в база данни, който да даде тайната, с която да се декриптира и валидира подписа.

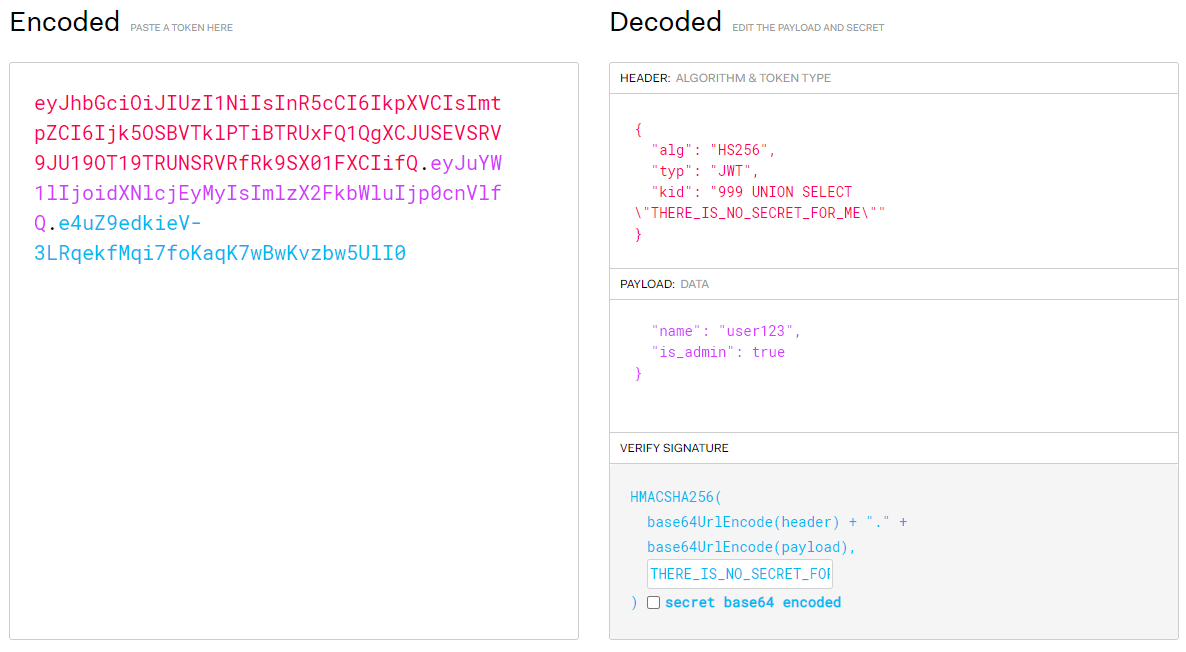
За да получим нужния достъп се изисква is\_admin да стане true и заявката да бъде верифицирана. За целта първо се подправя тялото да съдържа is\_admin: true и да се замести с тайна която е измислена от нас – например нека тайната е THERE\_IS\_NO\_SECRET\_FOR\_ME.

За целта kid трябва да замени това от базата, което ще стане с sql инжекция както следва:

999 UNION SELECT ‘THERE\_IS\_NO\_SECRET\_FOR\_ME’

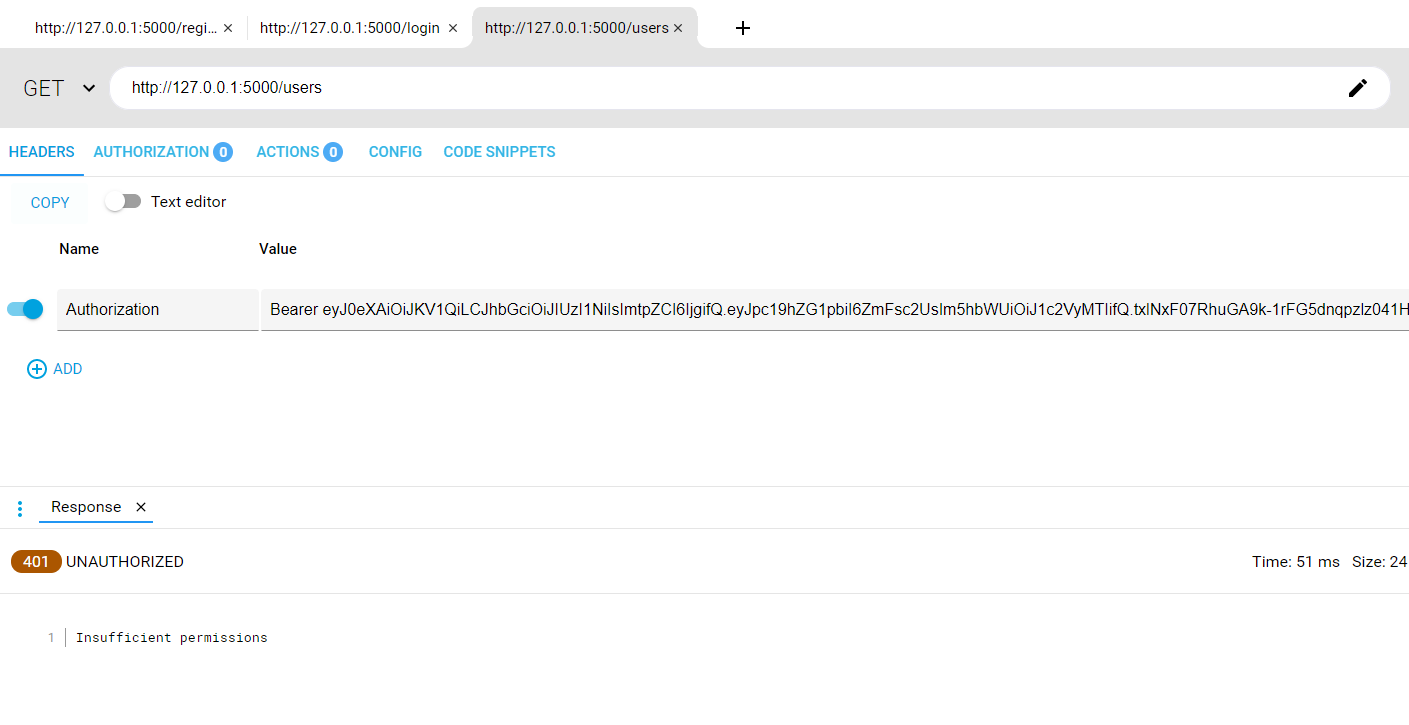
Ако случайно се окаже, че съществува ключ с идентификатор 999 се повтаря заявката докато не се намери несъществуващ в базата ключ.

С данните се създава нов JSON Web Token (Фигура 3), който има вида:

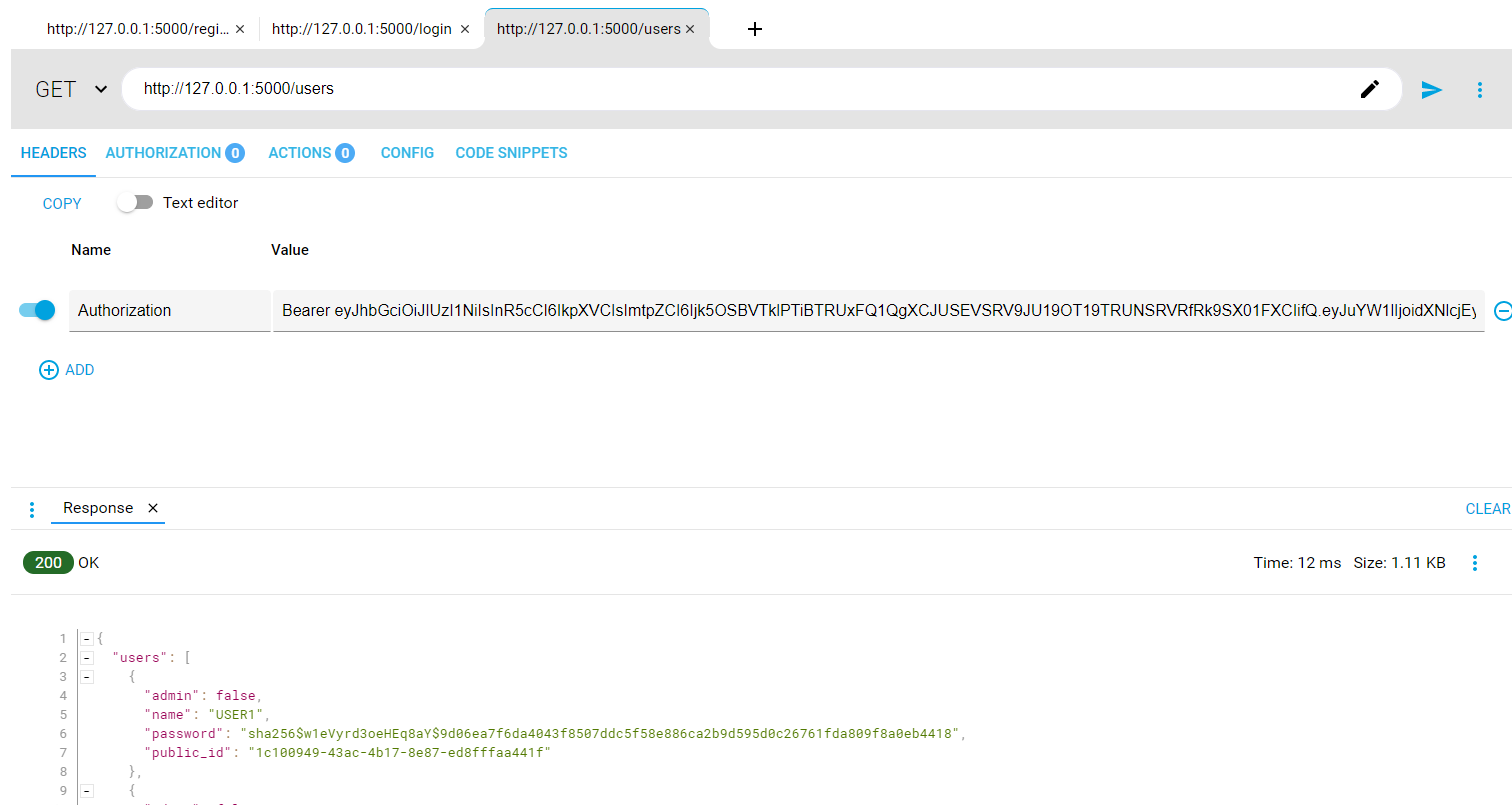
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ijk5OSBVTklPTiBTRUxFQ1QgXCJUSEVSRV9JU19OT19TRUNSRVRfRk9SX01FXCIifQ.eyJuYW1lIjoidXNlcjEyMyIsImlzX2FkbWluIjp0cnVlfQ.e4uZ9edkieV-3LRqekfMqi7foKaqK7wBwKvzbw5UlI0 

Фигура 3 кодиране на злонамерен токън [1]

С ново генерирания токън, вече може да се достъпи ограничения ресурс. Върнат резултат с оригиналния върнат от сървъра токън може да се види на фигура 4. На фигура 5 може да се види как сървърът използва дадената от SQL инжекцията тайна THERE\_IS\_NO\_SECRET\_FOR\_ME и съответно се връща резултат от дадения ресурс.



Фигура 4 резултат с оригиналния токън, който няма права за дадения ресурс[6]



Фигура 5 резултат със злонамерен токън[6]

Подхода за защита от този вид атаки е същия като при всички атаки обвързани с инжекции. Трябва стойностите на kid, да се валидират и в никакъв случай да не се използват директно във вида, в който идват. Конкретно за случая на SQL инжекции, трябва да се използват ORM-и и/или prepared statements, процедури и функции.

От по-горния пример проблемен код за взимане на тайната от базата има вида:

def **get\_secret\_from\_db**(kid):

    result = db.engine.execute("SELECT secret FROM Jwt Where Id = " + kid)

    secret = [row[0] for row in result][0]

    return secret

Той трябва да бъде заменен с:

def **get\_secret\_from\_db\_secure**(kid):

    return **Jwt**.query.filter\_by(id=kid).first().secret

### Атаки използващи jku

В заглавната част на JSON Web Token-а има параметър jku (JSON Web key Set Url) идеята на този параметър е сходна на тази на KID, този параметър посочва адрес, на който може да бъде достъпен файл – най-често във формат json, в който се съдържа публичен ключ, като част от алгоритъма за подписване.

Отново за този вид атака се започва от декодиране на получения от сървъра токен.

{

"alg": "RS256",

"typ": "JWT",

"jku":"https://example.com/key.json"

}.

{

"name": "user12",

"is\_admin": false

}

Като на адреса <https://example.com/key.json> има файл от типа:

{

"kty": "RSA",

"n": "-4KIwb83vQMH0YrzE44HppWvyNYmyuznuZPKWFt3e0xmdi-WcgiQZ1TC...RMxYC9lr4ZDp-M0",

"e": "AQAB"}

Виждайки нещо такова е ясно, че алгоритъмът за валидация на съобщения се състои от стъпки описани на фигура 6.



Фигура 6 Алгоритъм на работа при използване на JKU

От картинката се вижда как практически, ако се реферира файл зададен от хакера, може да създаден свой собствен подпис, който да бъде валиден за използвания файл.

За да се защитят системите от този вид атаки трябва системата задължително да има списък с позволени домейни, от които могат да се четат файлове. Също важно е да се използва DNS йерерахия на имената, да се забранят препращания (redirect-и). [3]

# Заключение

JSON Web Token-ите предлагат сигурен начин за проверка на самоличността и ограничаване на достъпа до ресурси. Като цяло проблемите, които се срещат са лесно отстраними и при правилната имплементация и експлоатация на този стандарт се гарантира висока надеждност. Разбира се винаги трябва да се валидира сигурността на направеното приложение с инструменти като jwt\_tool и други.[5]

# Демонстрация

За демонстрация се използва python проект разработен с рамката Flask, ORM – Flash\_SQLAlchemy и pyjwt.

Проектът е качен в хранилище за код в GitHub на този адрес – <https://github.com/ivanmilevtues/jwt-vulnariabilities-demo>

За да се създаде нужната среда с нейните депендънсите се пуска следната команда от главната директория на проекта:

conda env create –f jwt-demo.yml

За да се пусне проекта и да се изпробва позволен алгоритъм None се пуска с:

python app.py verify

За да се пусне проекта така, че да има възможност за SQL Injection през kid се пуска с:

python app.py kid

# Използвана литература

[1] JWT auth0, <https://jwt.io>

- уебсайт

[2] JSON Web Token (JWT), <https://datatracker.ietf.org/doc/html/rfc7519>

- описание на стандарт

[3] Web Security Blog, Timur Güvenkaya, <https://www.invicti.com/blog/web-security/json-web-token-jwt-attacks-vulnerabilities/>

- уебсайт

[4] HackTrics, <https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens>

- уебсайт

[5] Attack Methodology, <https://github.com/ticarpi/jwt_tool/wiki/Attack-Methodology>

- уебсайт

[6] Advacned Rest Client, <https://install.advancedrestclient.com/install>

- инструмент