**API testing**

APIs (Application Programming Interfaces) enable software systems and applications to communicate and share data. API testing is important as vulnerabilities in APIs may undermine core aspects of a website's confidentiality, integrity, and availability.

All dynamic websites are composed of APIs, so classic web vulnerabilities like SQL injection could be classed as API testing. In this topic, we'll teach you how to test APIs that aren't fully used by the website front-end, with a focus on RESTful and JSON APIs. We'll also teach you how to test for server-side parameter pollution vulnerabilities that may impact internal APIs.



**API recon**

To start API testing, you first need to find out as much information about the API as possible, to discover its attack surface.

To begin, you should identify API endpoints. These are locations where an API receives requests about a specific resource on its server. For example, consider the following GET request:

GET /api/books HTTP/1.1

Host: example.com

The API endpoint for this request is /api/books. This results in an interaction with the API to retrieve a list of books from a library.

Another API endpoint might be, for example,

GET /api/books/mystery HTTP/1.1

Host: example.com

/api/books/mystery, which would retrieve a list of mystery books.

Once you have identified the endpoints, you need to determine how to interact with them. This enables you to construct valid HTTP requests to test the API. For example, you should find out information about the following:

* The input data the API processes, including both compulsory and optional parameters.
* The types of requests the API accepts, including supported HTTP methods and media formats.
* Rate limits and authentication mechanisms.

**API documentation**

APIs are usually documented so that developers know how to use and integrate with them.

Documentation can be in both human-readable and machine-readable forms.

Human-readable documentation is designed for developers to understand how to use the API. It may include detailed explanations, examples, and usage scenarios.

Machine-readable documentation is designed to be processed by software for automating tasks like API integration and validation. It's written in structured formats like JSON or XML.

API documentation is often publicly available, particularly if the API is intended for use by external developers. If this is the case, always start your recon by reviewing the documentation.

Discovering API documentation

Even if API documentation isn't openly available, you may still be able to access it by browsing applications that use the API.

To do this, you can use Burp Scanner to crawl the API. You can also browse applications manually using Burp's browser. Look for endpoints that may refer to API documentation, for example.

* /api
* /swagger/index.html
* /openapi.json

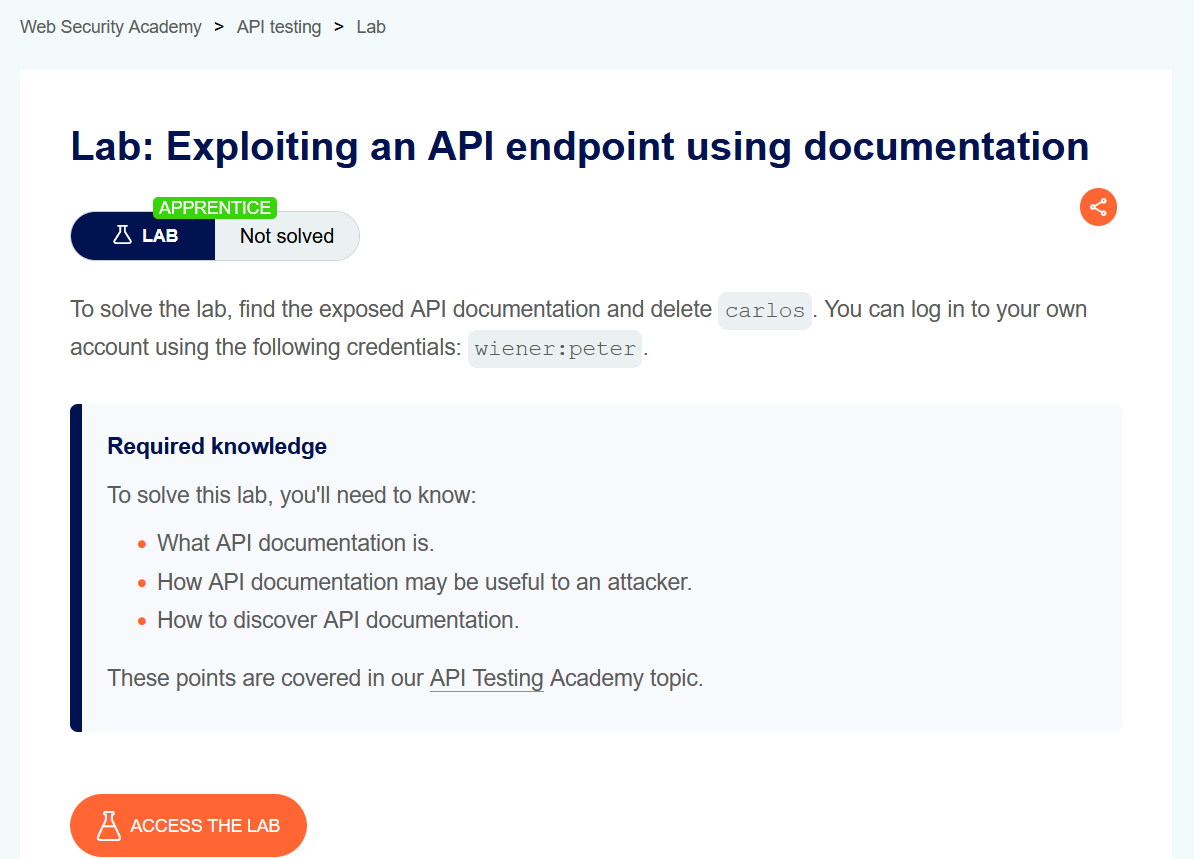
If you identify an endpoint for a resource, make sure to investigate the base path. For example, if you identify the resource endpoint /api/swagger/v1/users/123, then you should investigate the following paths:

* /api/swagger/v1
* /api/swagger
* /api

We can also use a list of common paths to find documentation using Intruder.

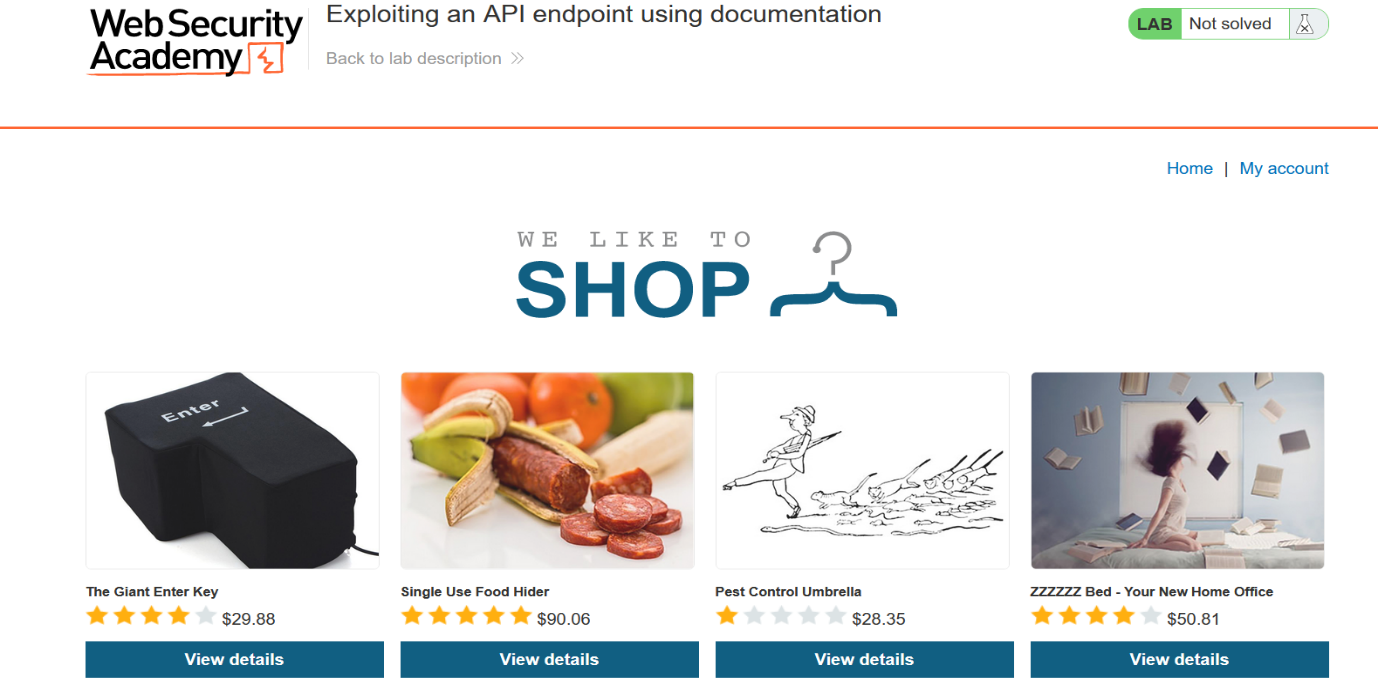
# Lab: 1. Exploiting an API endpoint using documentation.

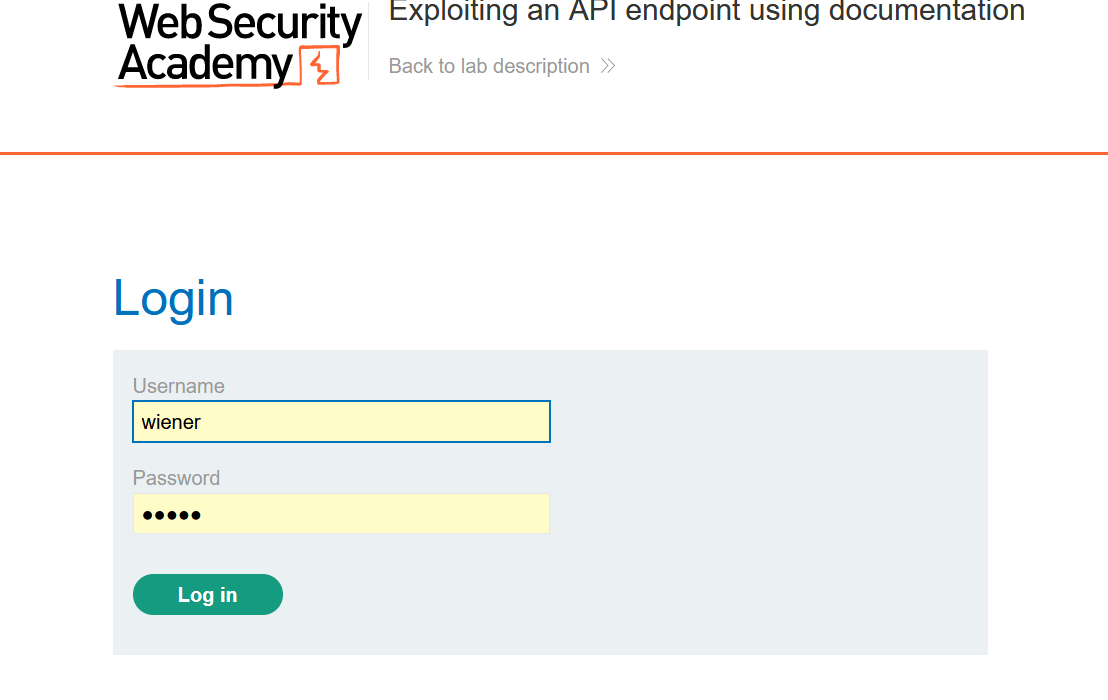
<https://portswigger.net/web-security/api-testing/lab-exploiting-api-endpoint-using-documentation>



In this lab scenario we need to delete user Carlos, and we have our own credential as user=wiener, pass=peter.

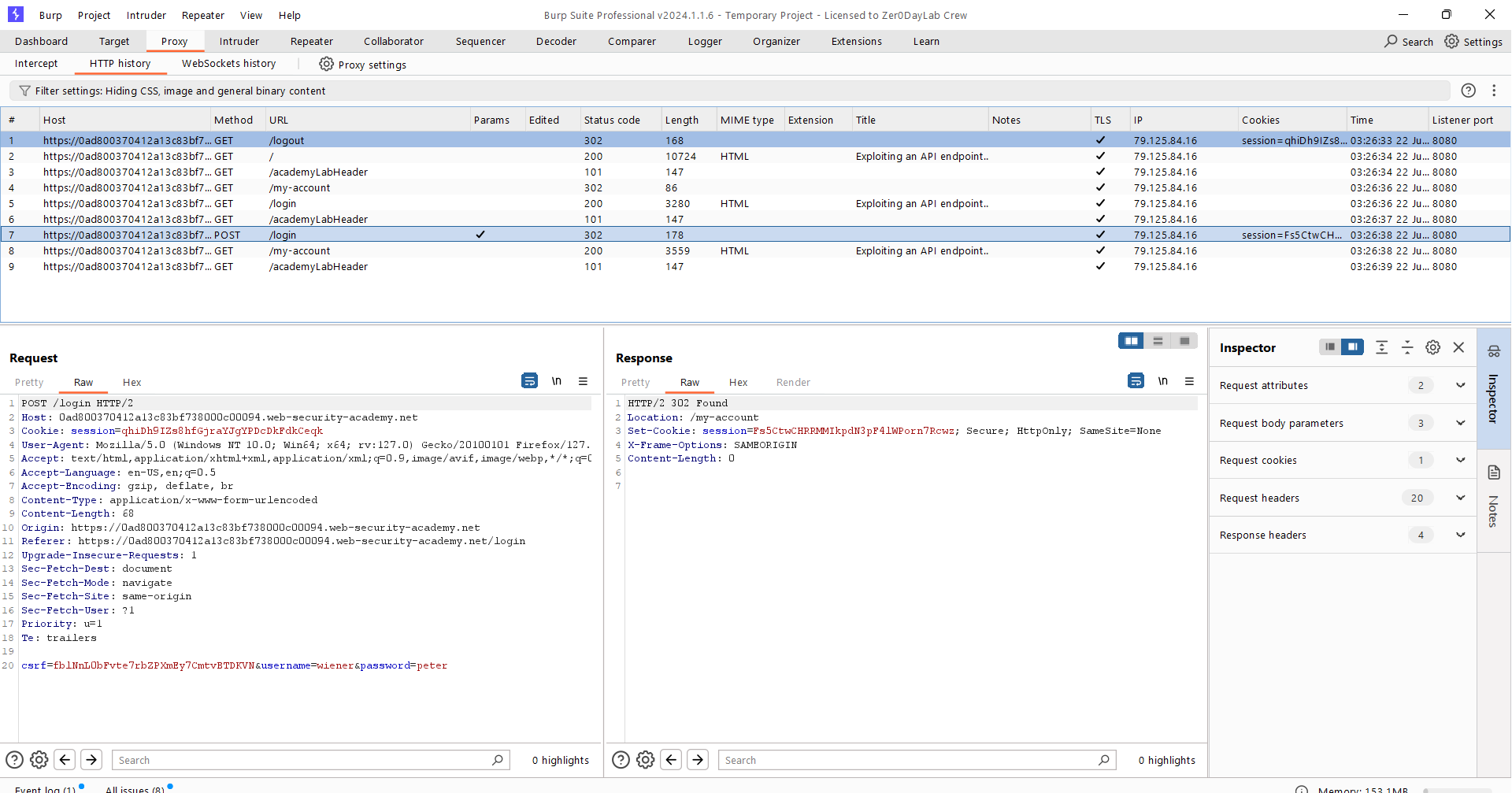
**Let access the lab**

****

**To My account**

**Let login**

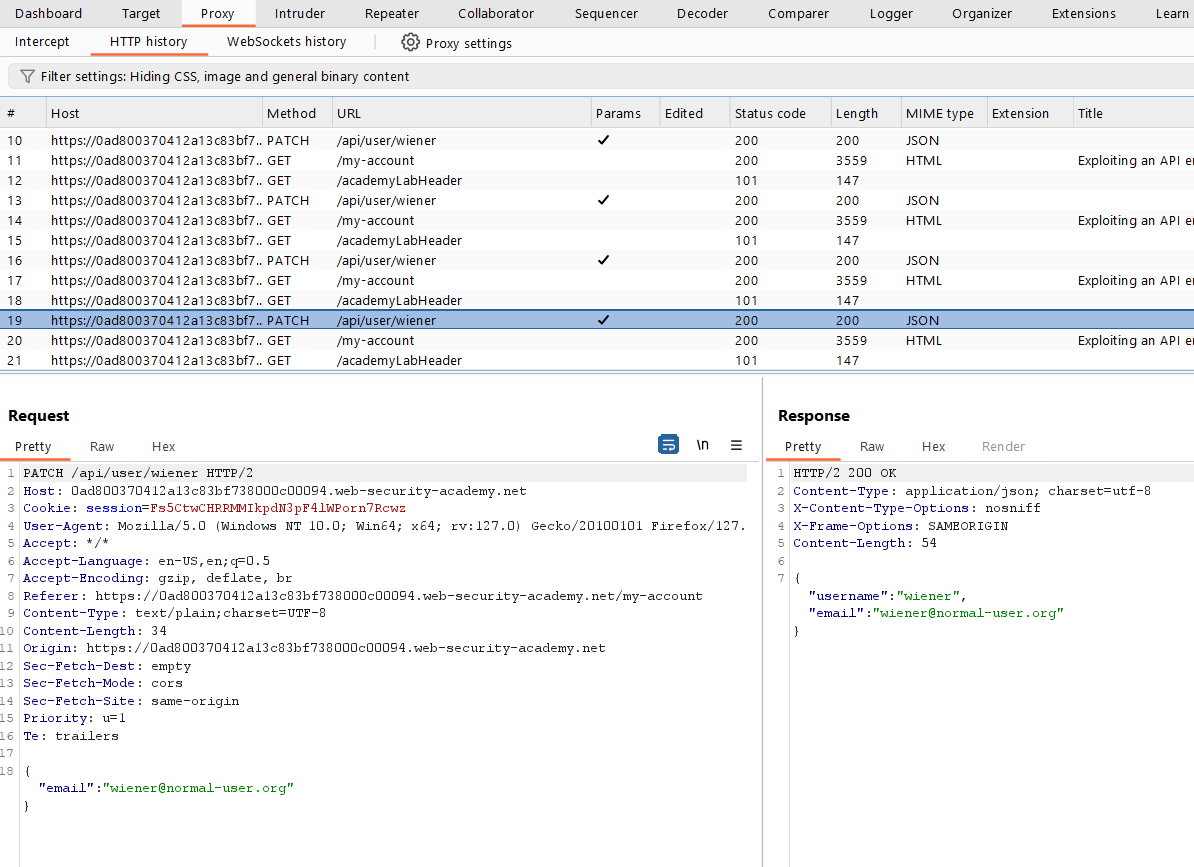
**If we saw our burp history. We could find anything so , we need to interact with api .**

****

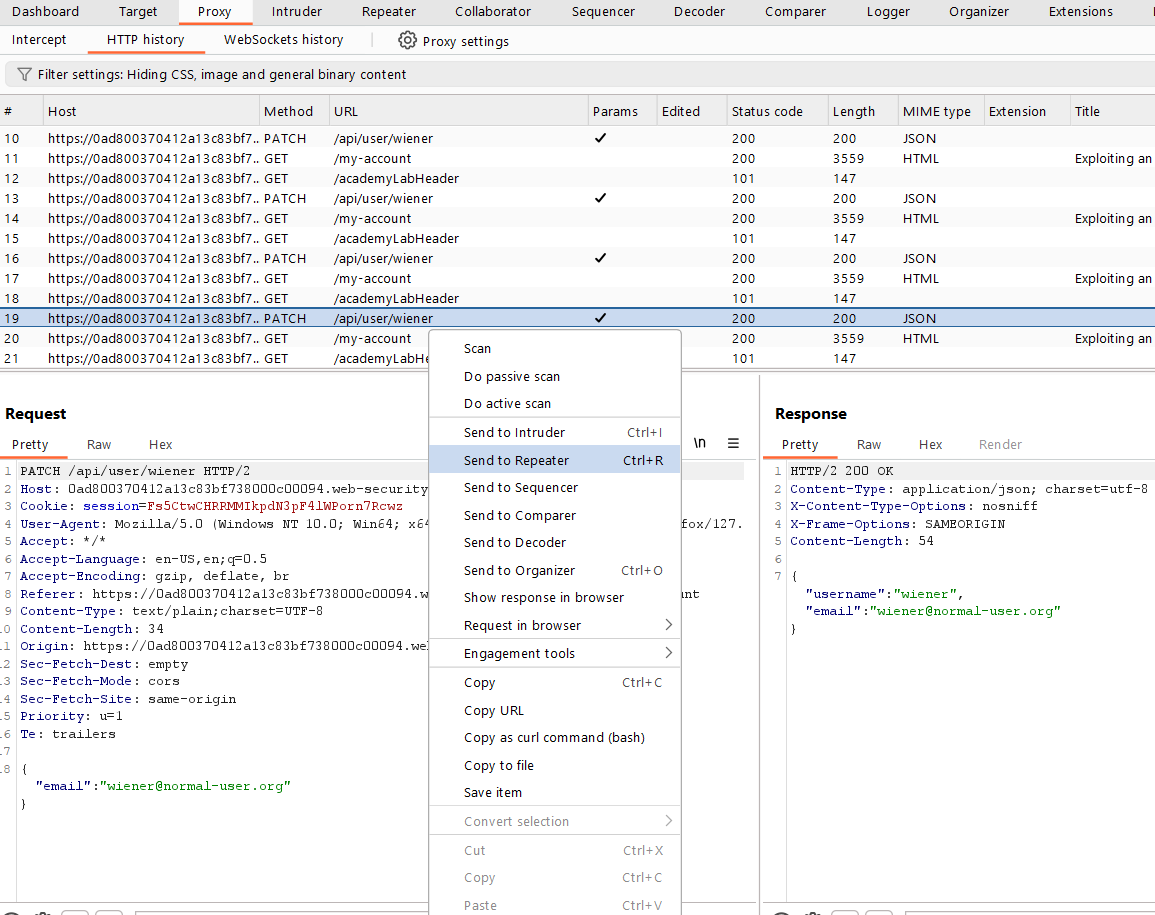
**After we got log-in we need to update our mail to interact with api**

****

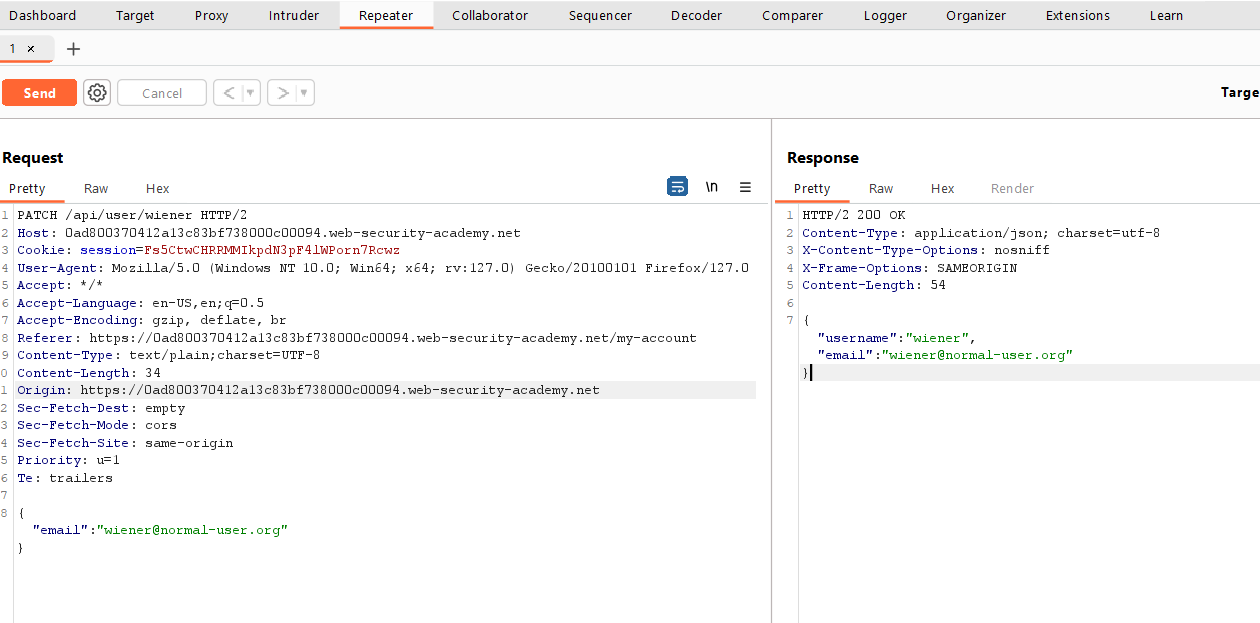
**As I update my mail, I got a Patch request**

****

**Let send this to repeater**

****

**In repeater send this request and see response**

****

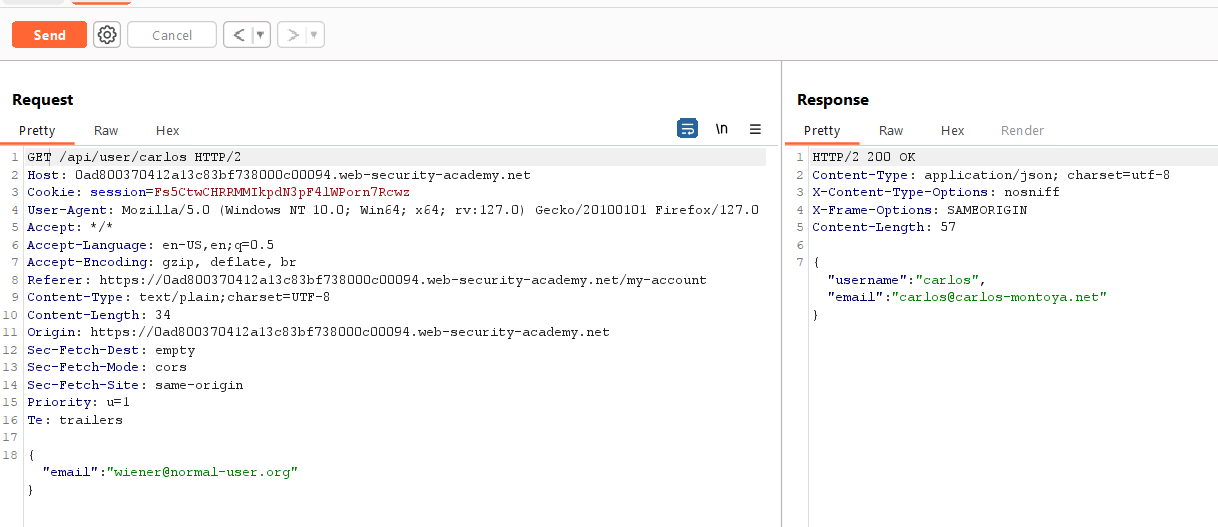
**We can able to see our username and mail.**

**I try to change the api path /api/user/wiener => /api/user/carlos**

****

**But I got this 400**

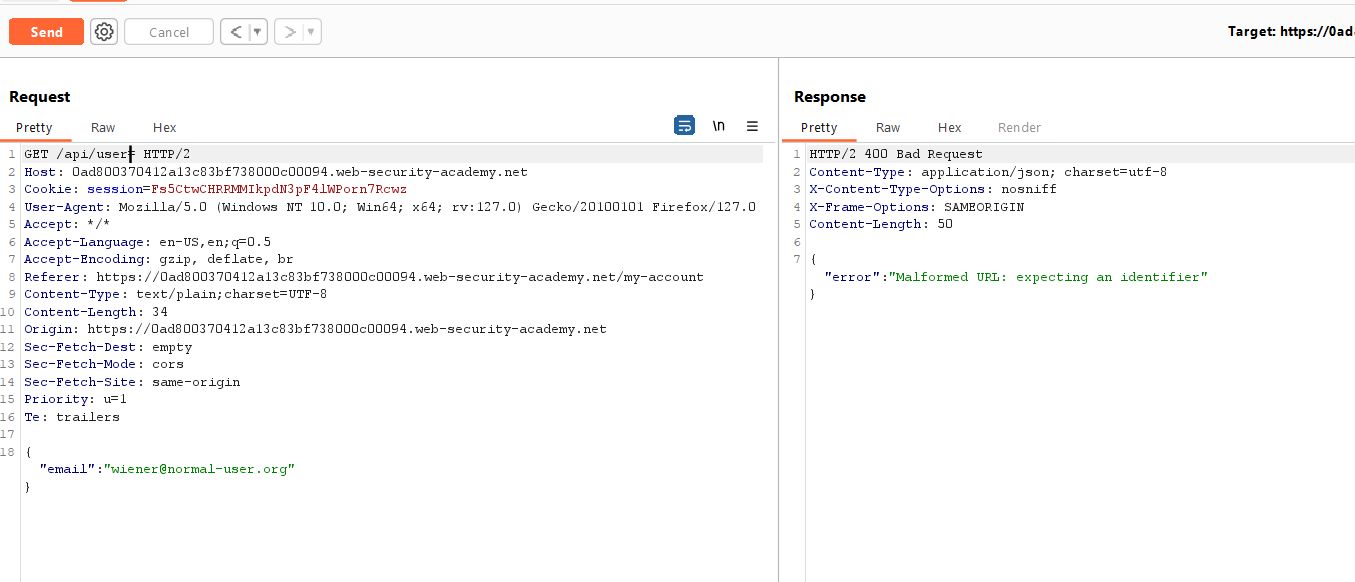
**So let change PATCH to GET**

****

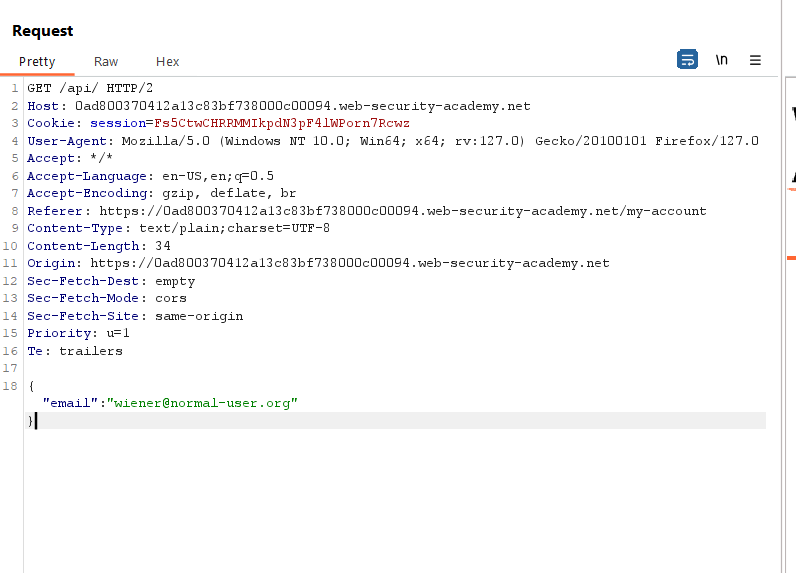
**I got Carlos details**

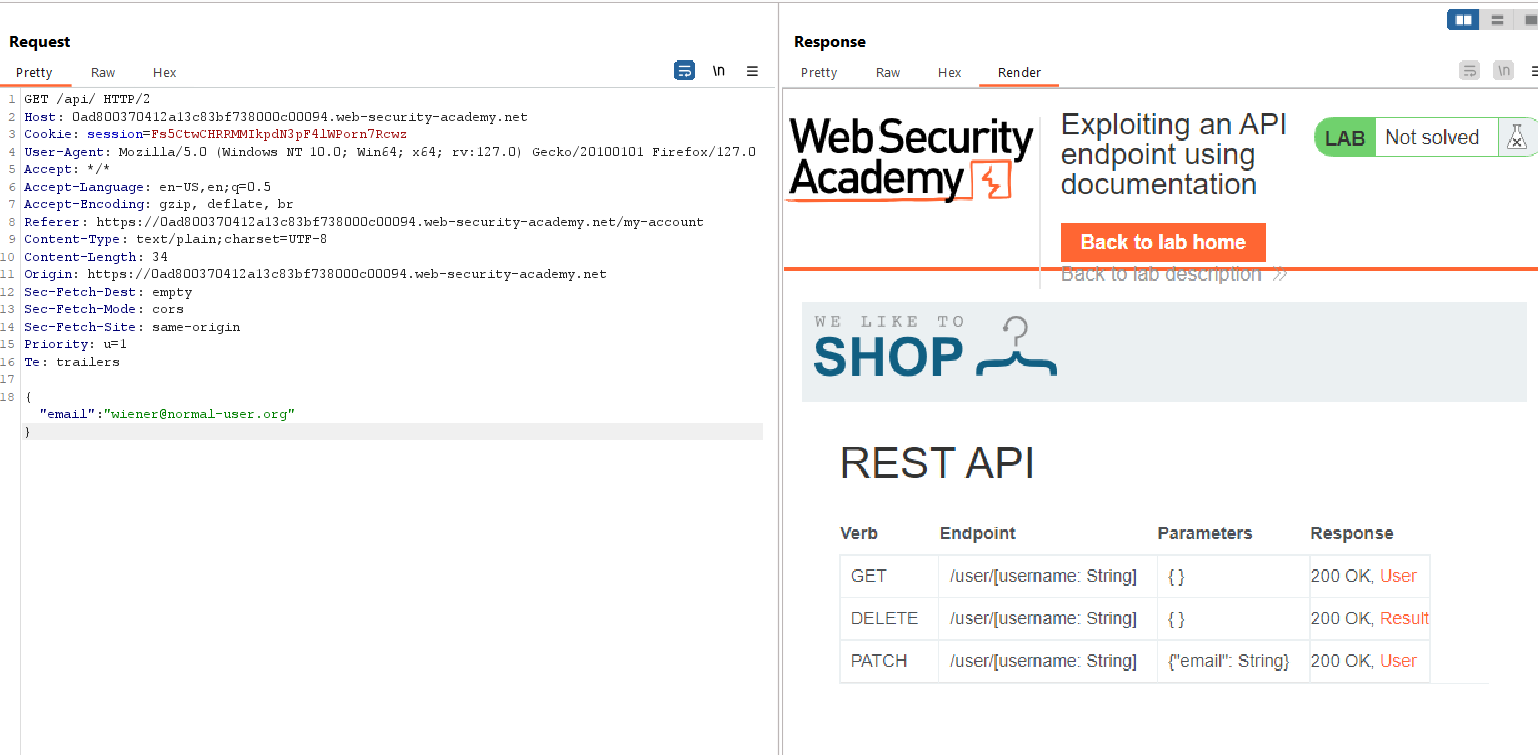
**So, we can delete from here but before doing it Let check some endpoint so let just remove Carlos.**

**Can I see all user details**

****

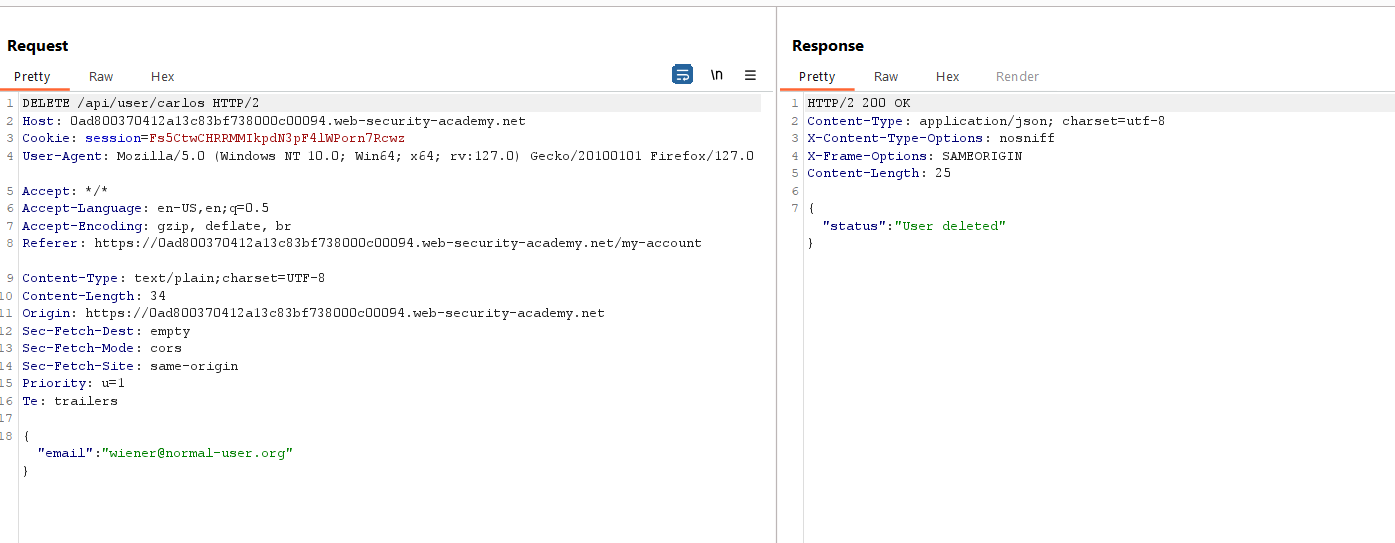
**That unexpected but I got 400 again. So let just remove /user parameter as well.**

****

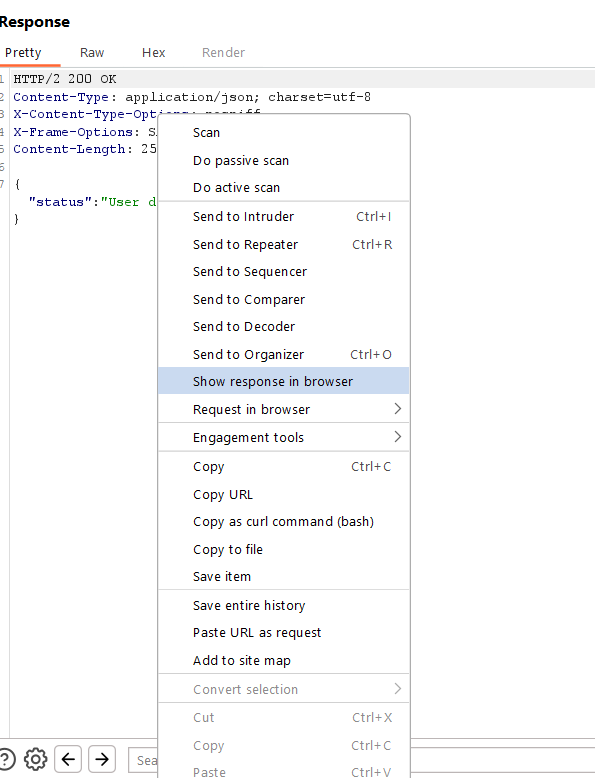
****

**OH, so here are methods that are allowed**

**Let just delete and end this lab there is nothing more interesting.**

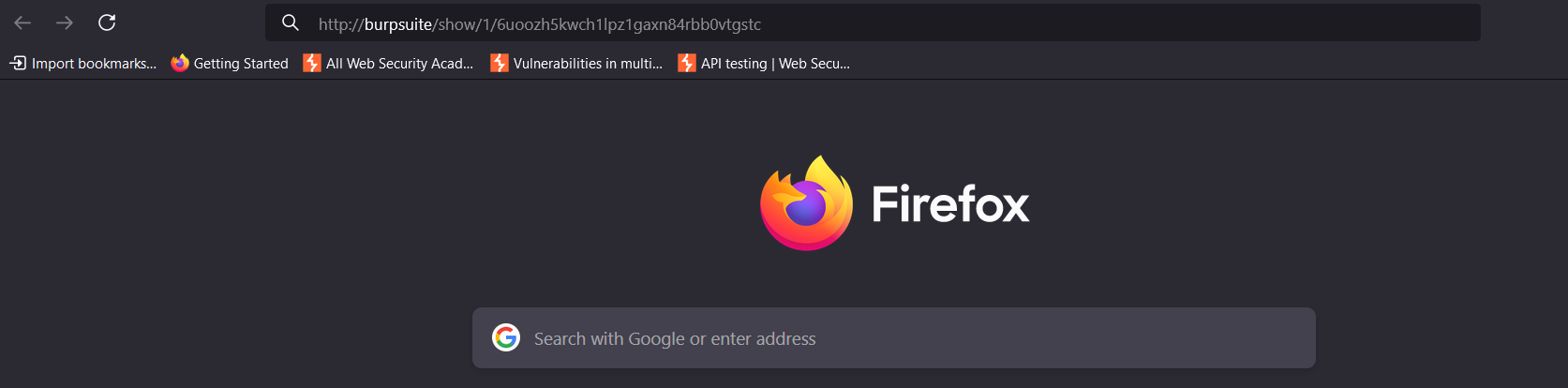
**So, add DELETE in place of GET and user name Carlos.**

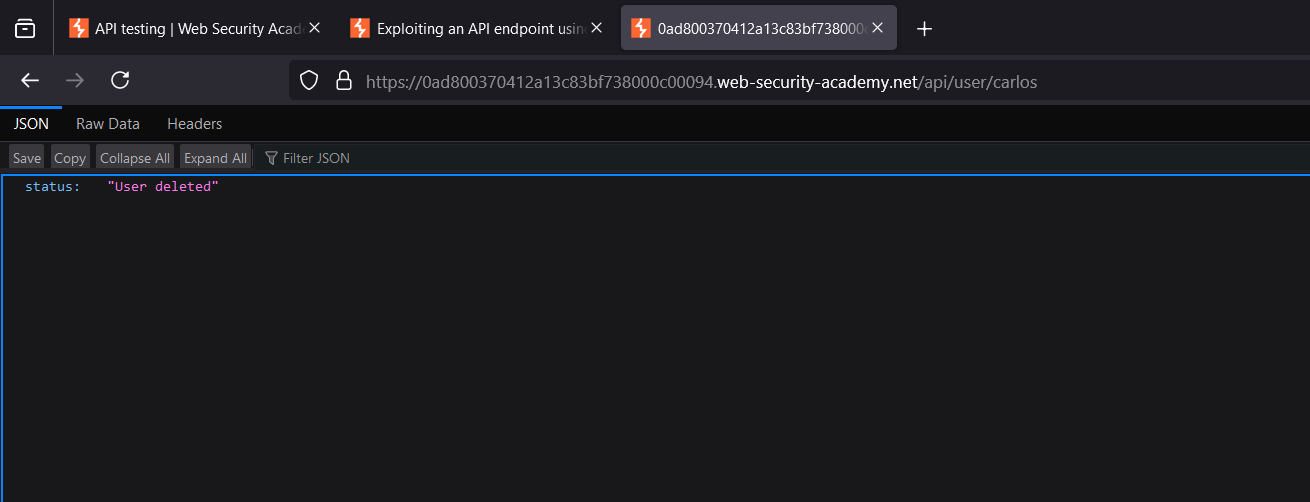
**Let show this response in browser.**

****

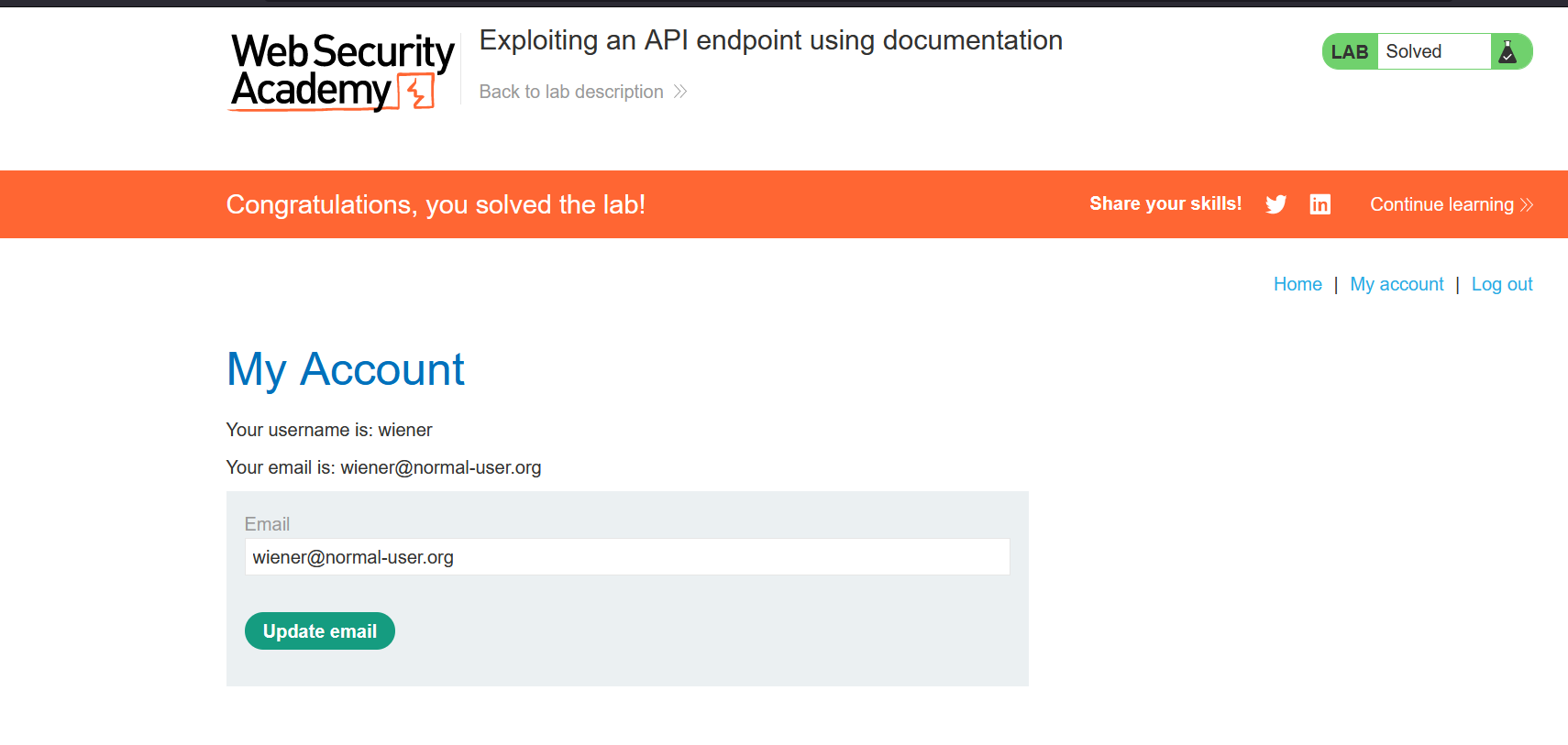
**Copy URL and past in browser.**

****

****

****

**And User is Deleted**

****

**Done**

# Using machine-readable documentation

You can use a range of automated tools to analyze any machine-readable API documentation that you find.

You can use Burp Scanner to crawl and audit OpenAPI documentation, or any other documentation in JSON or YAML format. You can also parse OpenAPI documentation using the [OpenAPI Parser](https://portswigger.net/bappstore/6bf7574b632847faaaa4eb5e42f1757c) BApp.

You may also be able to use a specialized tool to test the documented endpoints, such as [Postman](https://www.postman.com/) or [SoapUI](https://www.soapui.org/).

**Identifying API endpoints**

You can also gather a lot of information by browsing applications that use the API. This is often worth doing even if you have access to API documentation, as sometimes documentation may be inaccurate or out of date.

You can use Burp Scanner to crawl the application, then manually investigate interesting attack surface using Burp's browser.

While browsing the application, look for patterns that suggest API endpoints in the URL structure, such as /api/. Also look out for JavaScript files. These can contain references to API endpoints that you haven't triggered directly via the web browser. Burp Scanner automatically extracts some endpoints during crawls, but for a more heavyweight extraction, use the [JS Link Finder](https://portswigger.net/bappstore/0e61c786db0c4ac787a08c4516d52ccf) BApp. You can also manually review JavaScript files in Burp.

Interacting with API endpoints

Once you've identified API endpoints, interact with them using Burp Repeater and Burp Intruder. This enables you to observe the API's behavior and discover additional attack surface. For example, you could investigate how the API responds to changing the HTTP method and media type.

As you interact with the API endpoints, review error messages and other responses closely. Sometimes these include information that you can use to construct a valid HTTP request.

**Identifying supported HTTP methods**

The HTTP method specifies the action to be performed on a resource. For example:

* GET - Retrieves data from a resource.
* PATCH - Applies partial changes to a resource.
* OPTIONS - Retrieves information on the types of request methods that can be used on a resource.

An API endpoint may support different HTTP methods. It's therefore important to test all potential methods when you're investigating API endpoints. This may enable you to identify additional endpoint functionality, opening up more attack surface.

For example, the endpoint /api/tasks may support the following methods:

* GET /api/tasks - Retrieves a list of tasks.
* POST /api/tasks - Creates a new task.
* DELETE /api/tasks/1 - Deletes a task.

You can use the built-in HTTP verbs list in Burp Intruder to automatically cycle through a range of methods.

Identifying supported content types

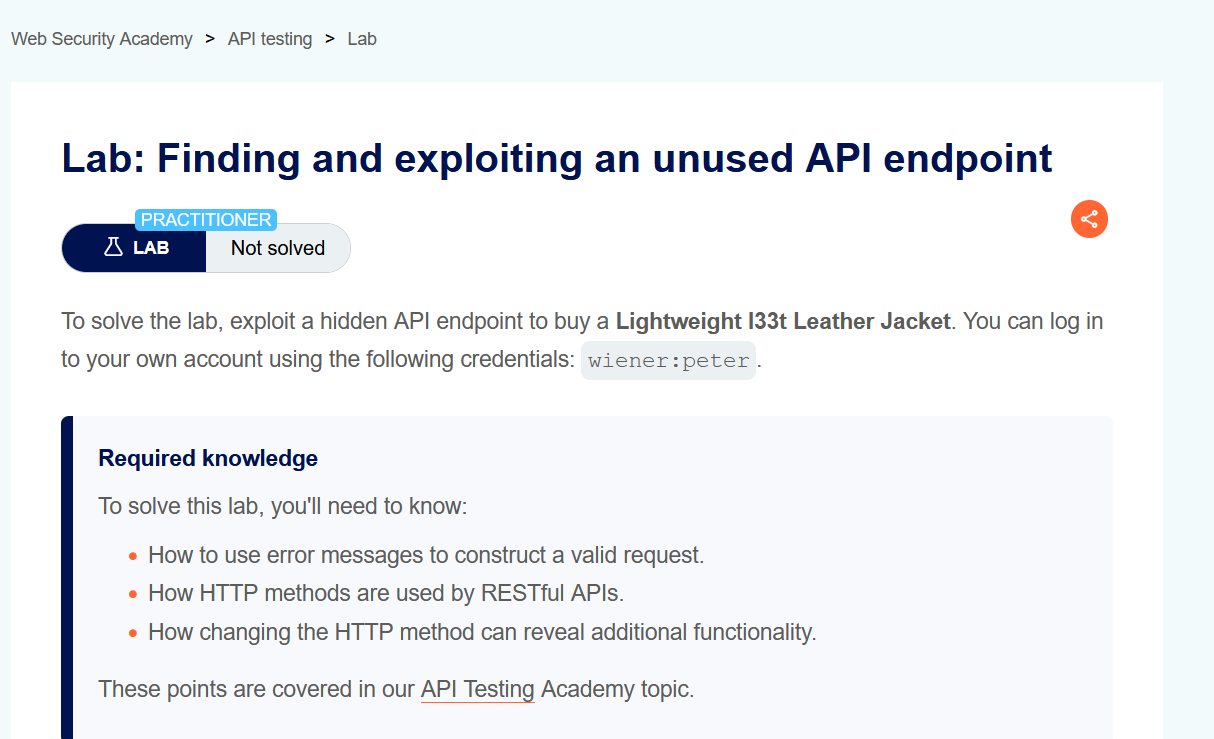
API endpoints often expect data in a specific format. They may therefore behave differently depending on the content type of the data provided in a request. Changing the content type may enable you to:

* Trigger errors that disclose useful information.
* Bypass flawed defences.
* Takes advantage of differences in processing logic. For example, an API may be secure when handling JSON data but susceptible to injection attacks when dealing with XML.

To change the content type, modify the **Content-Type header**, then reformat the request body accordingly. You can use the [Content type converter](https://portswigger.net/bappstore/db57ecbe2cb7446292a94aa6181c9278) BApp to automatically convert data submitted within requests between XML and JSON.

# Lab: 2. Finding and exploiting an unused API endpoint.

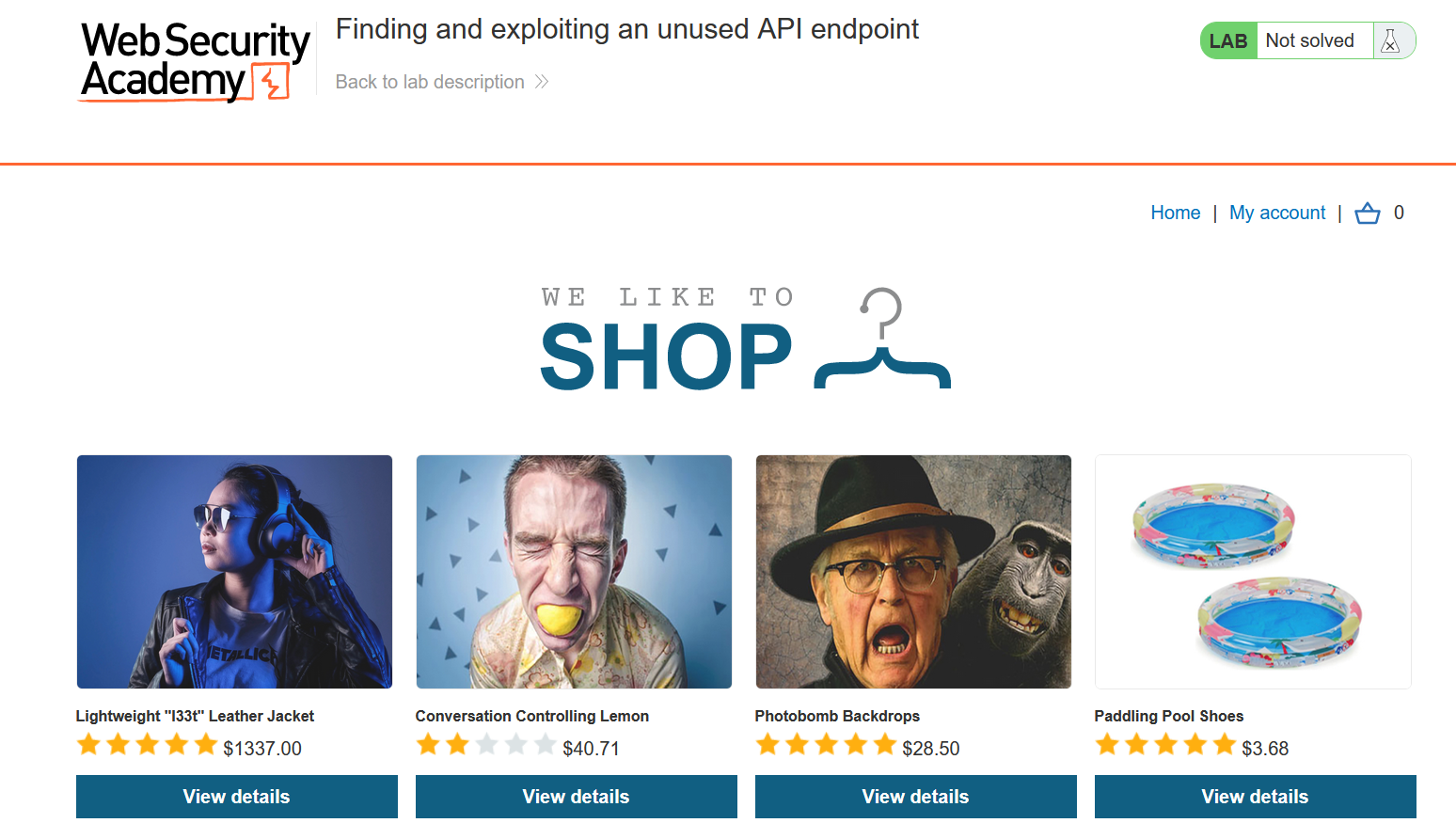
<https://portswigger.net/web-security/api-testing/lab-exploiting-unused-api-endpoint>



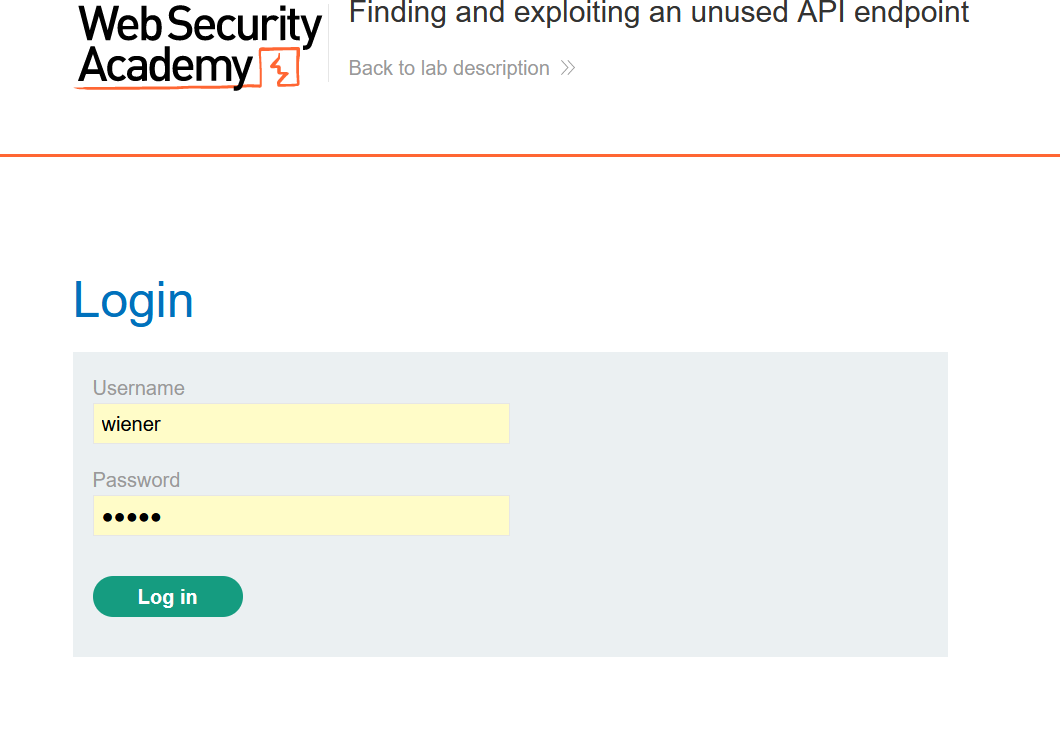
**In this lab scenario** we haver to buy a leather jacket. by using logical flow of API.

In order to solve this Lab, we need some basic knowledge of we methods.

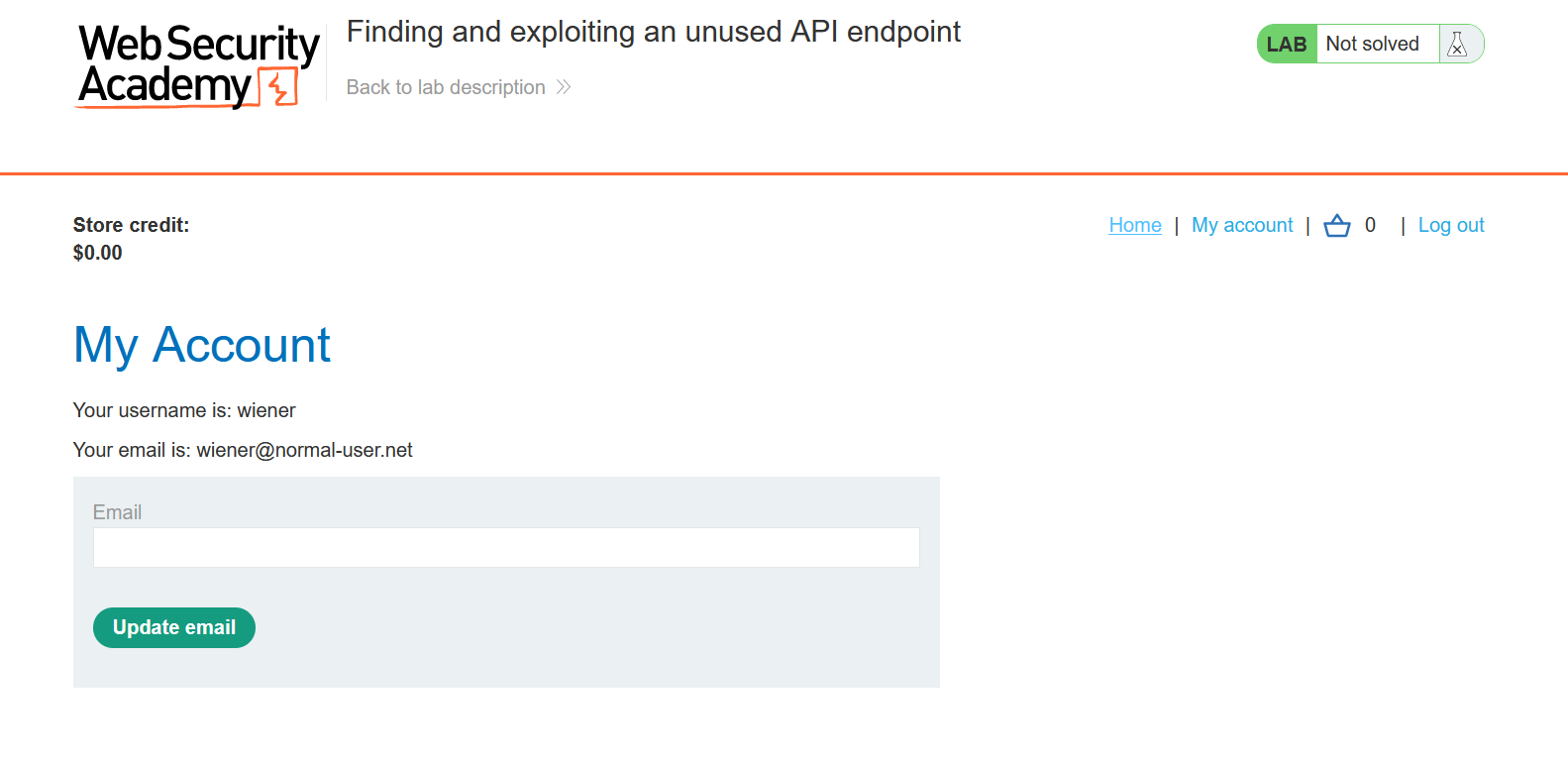
**Let access the lab**

****

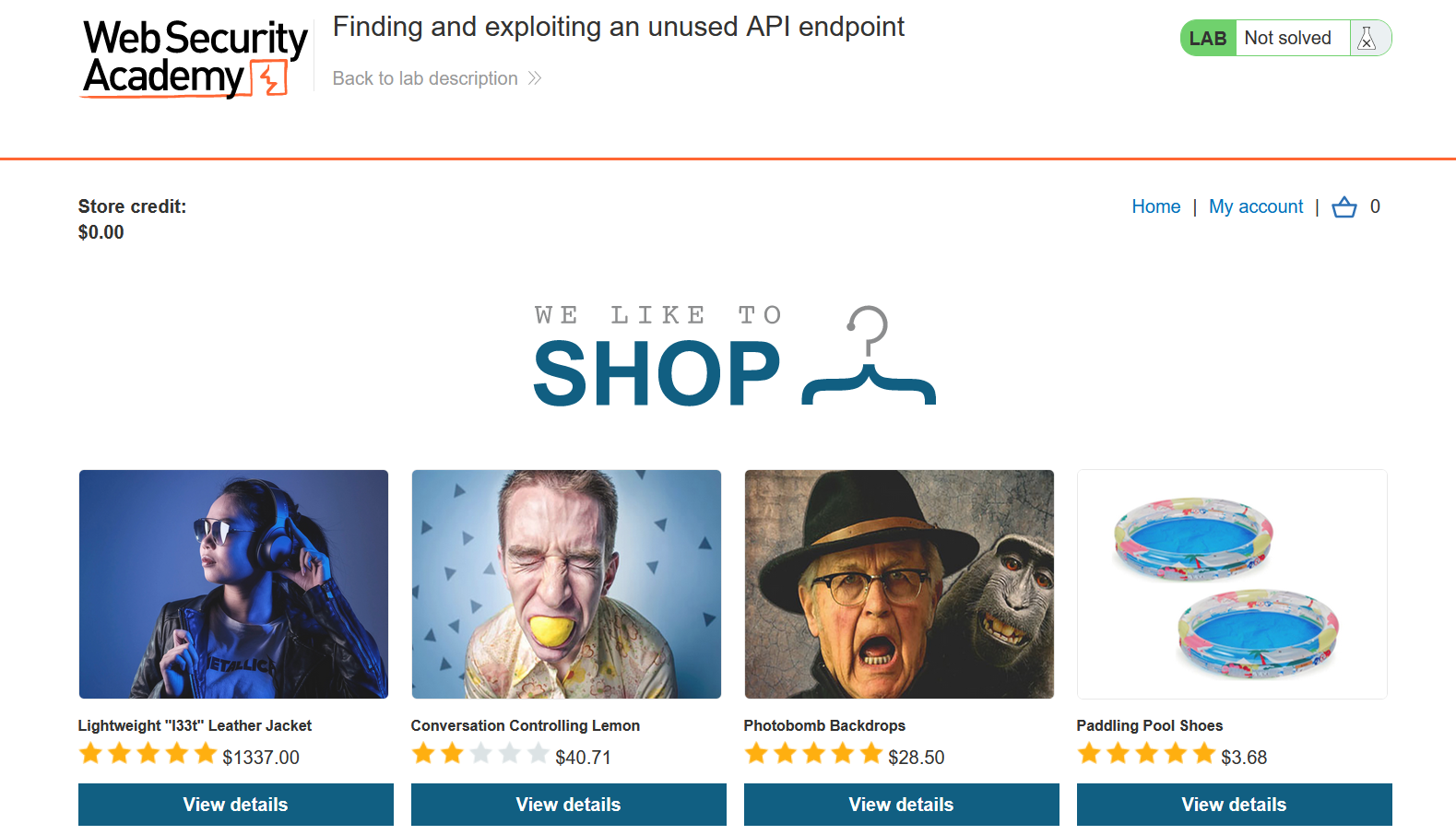
**Go to My account > login with given credential**

****

After login This page appear

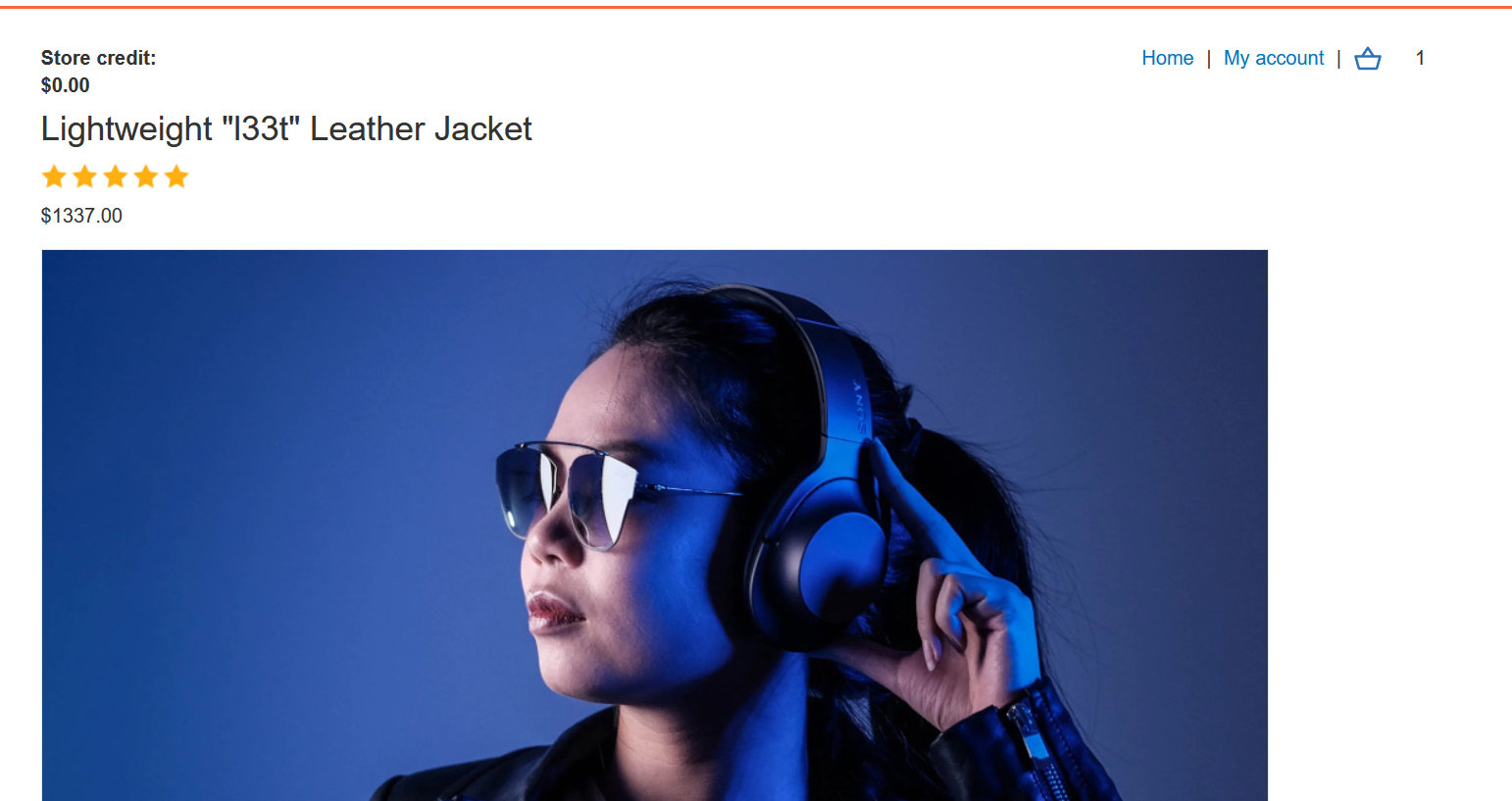


We have 0 credit > just go to home



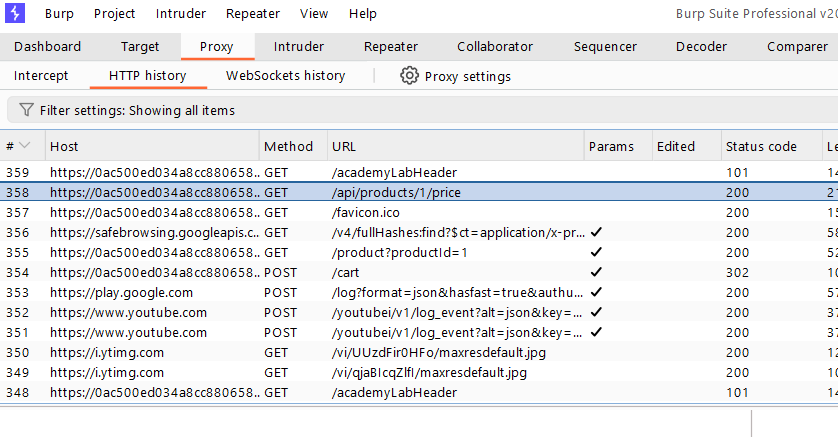
Let select the leather jacket

Let add this jacket in the cart

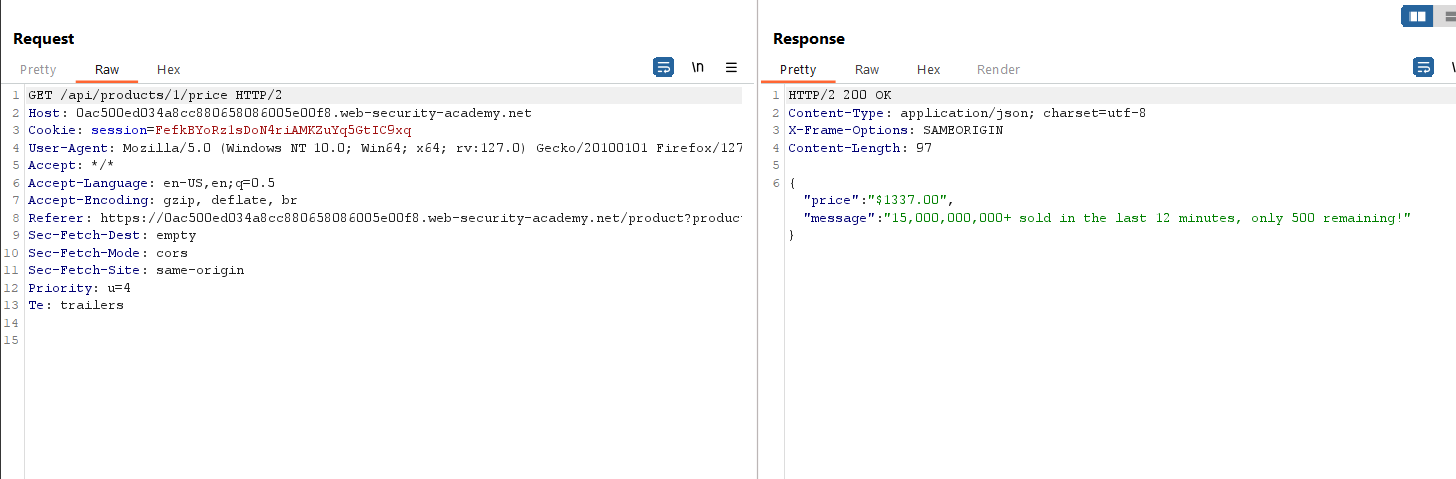


I just add this into my cart.

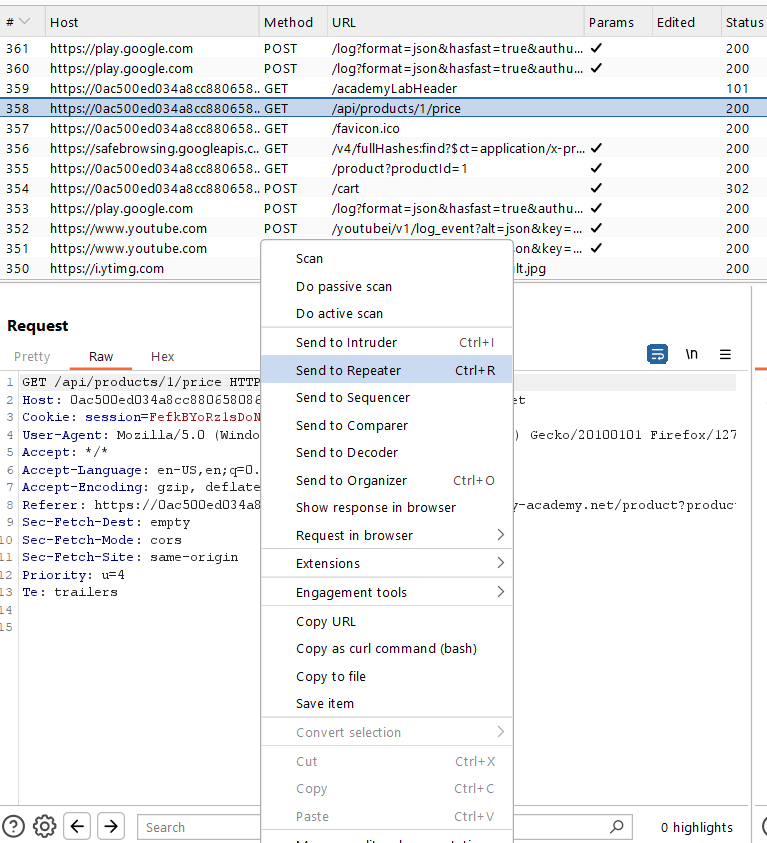
Let go to burp Http history, I get something interesting here.



Let check this request



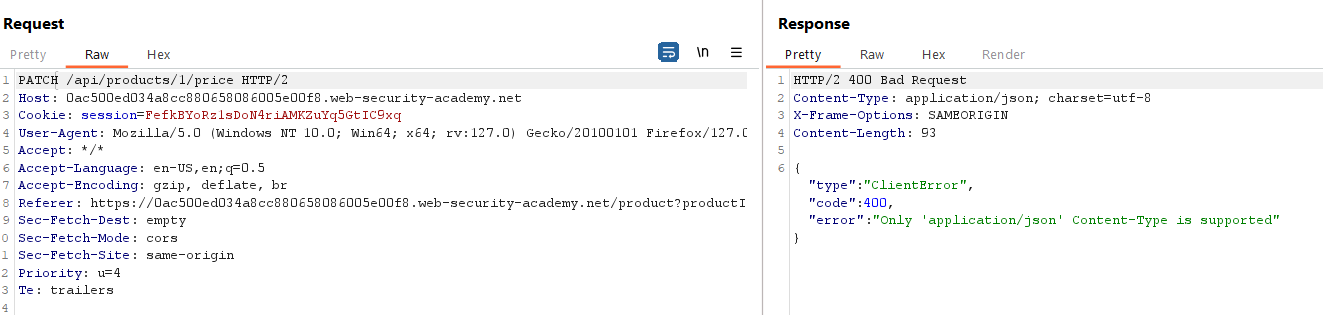
Let send this Request to Repeater to see more closely.



Let send this Request to see response



Let change Request type Get to Patch to observer changes



I got an error :

HTTP/2 400 Bad Request

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

X-Frame-Options: SAMEORIGIN

Content-Length: 93

{

"type":"ClientError",

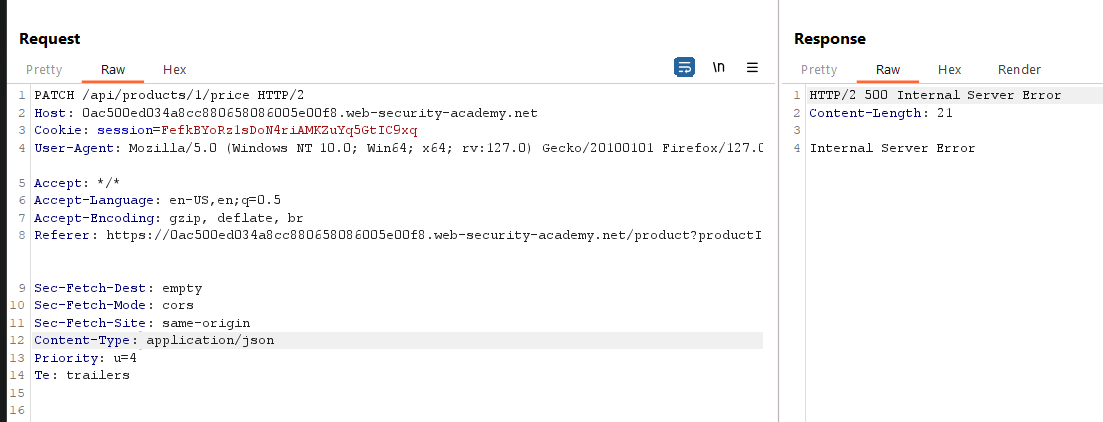
"code":400,

‘"error":"Only 'application/json' Content-Type is supported"

}

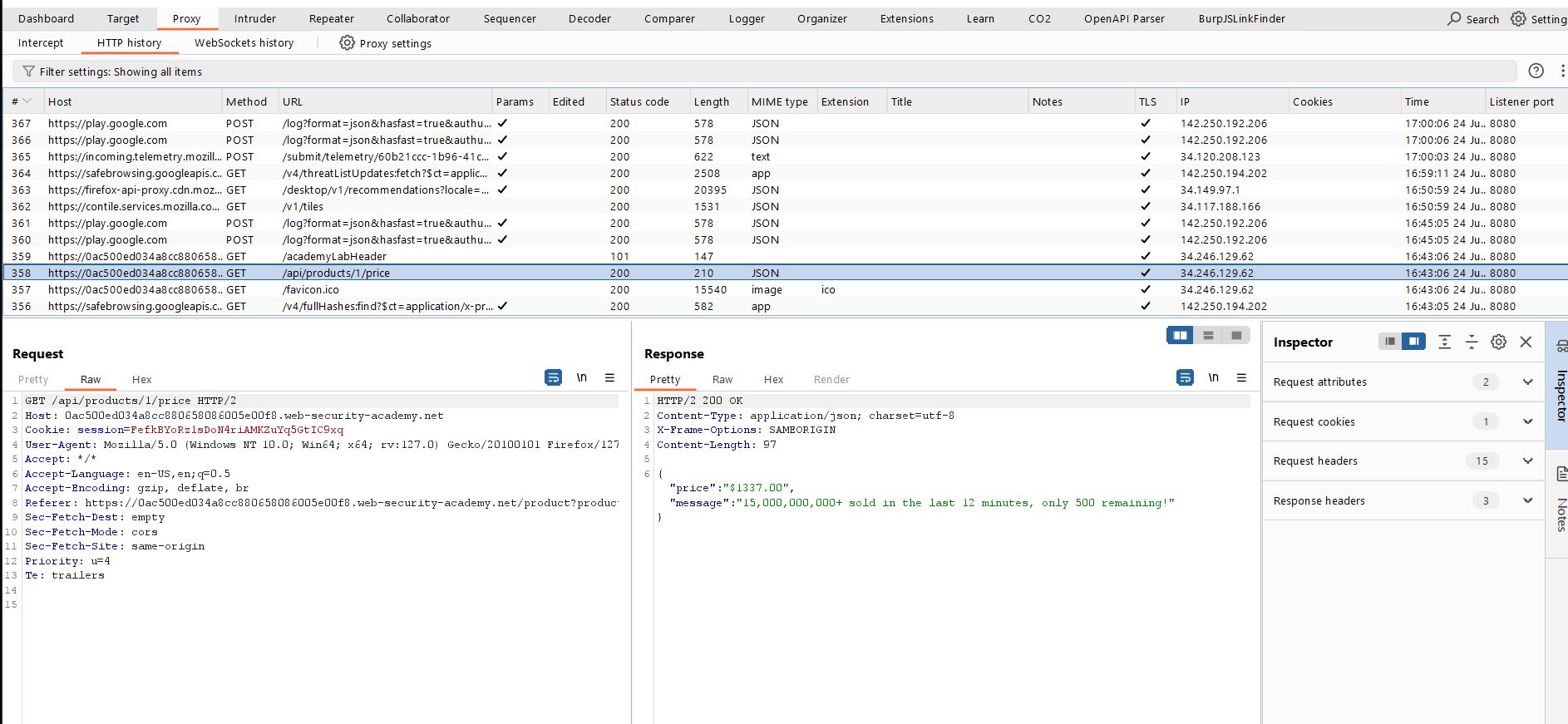
If we read it in upper line Content-Type is supported that mean we need to add Content-Type header in request and Content-Type support only application/Json.

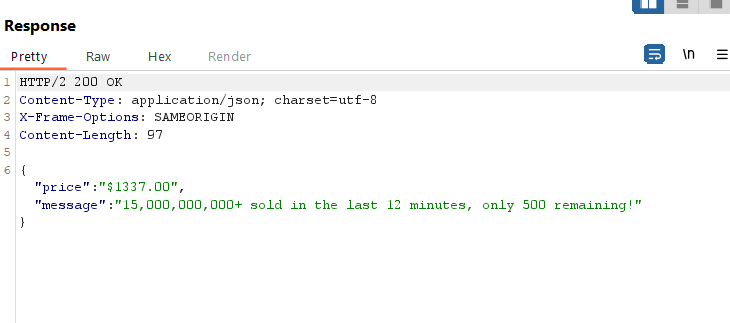
Let add in request



OK I’m getting bez I am using content-type header as Json but I also need to add in content length as I saw in previous request.

So, in order to do that just back to http history of burp and open same request.





Copy all this, OR copy from below. after that Modify price to 0.

Modified Request

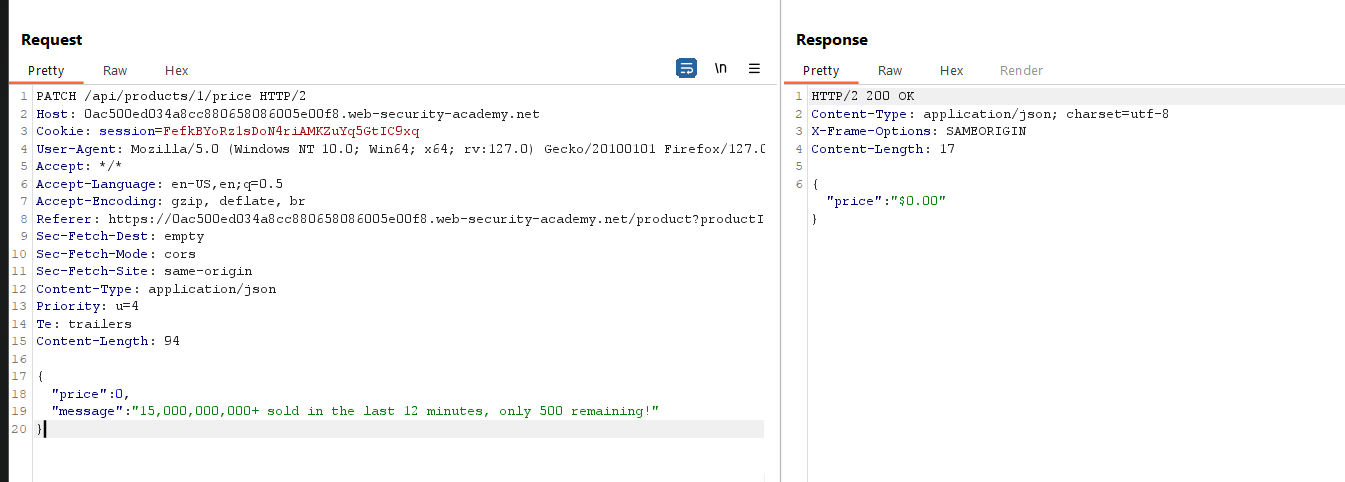
Content-Length: 97

{

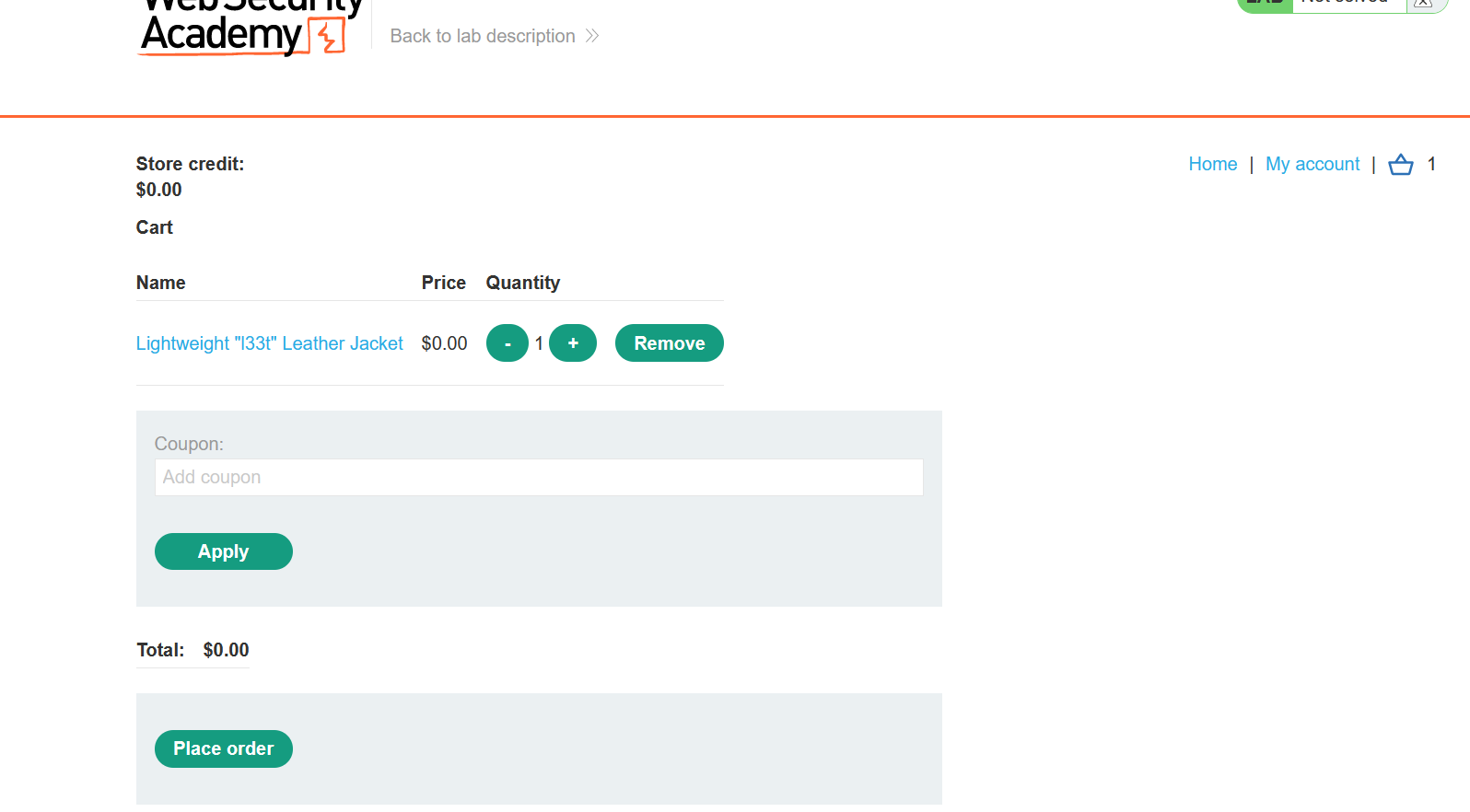
"price":0,

"message":"15,000,000,000+ sold in the last 12 minutes, only 500 remaining!"

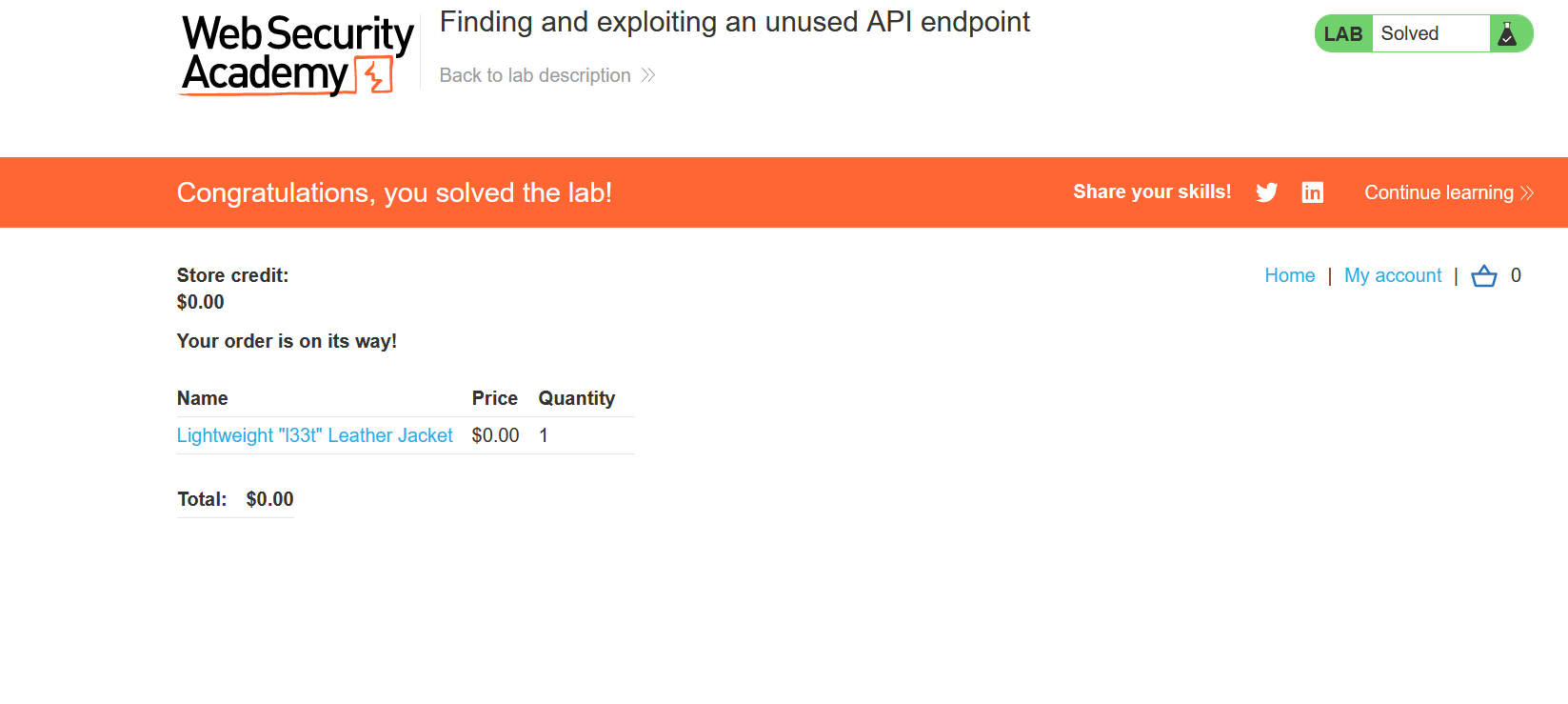
}



Done just open browser again, and go to cart



Done let place order



If u find any error in request

PATCH /api/products/1/price HTTP/2

Host: 0ac500ed034a8cc880658086005e00f8.web-security-academy.net

Cookie: session=FefkBYoRz1sDoN4riAMKZuYq5GtIC9xq

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:127.0) Gecko/20100101 Firefox/127.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

Referer: https://0ac500ed034a8cc880658086005e00f8.web-security-academy.net/product?productId=1

Sec-Fetch-Dest: empty

Sec-Fetch-Mode: cors

Sec-Fetch-Site: same-origin

Content-Type: application/json

Priority: u=4

Te: trailers

Content-Length: 94

{

"price":0,

"message":"15,000,000,000+ sold in the last 12 minutes, only 500 remaining!"

}

Here is my request, by the way we don’t need this message parameter.

Done…