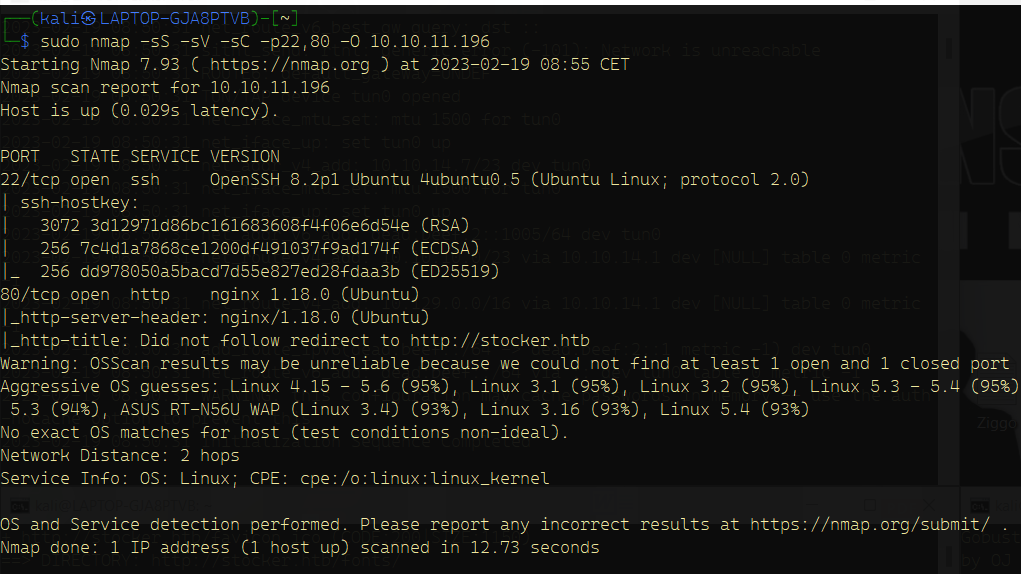
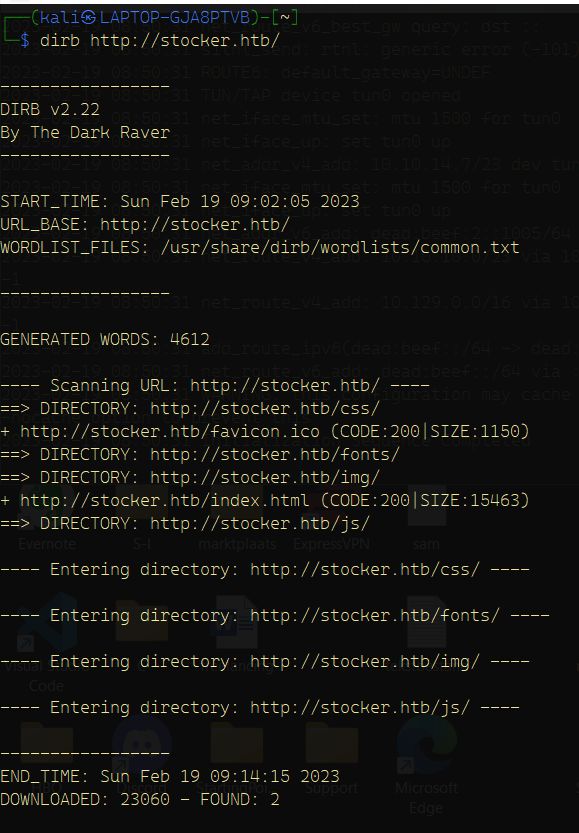
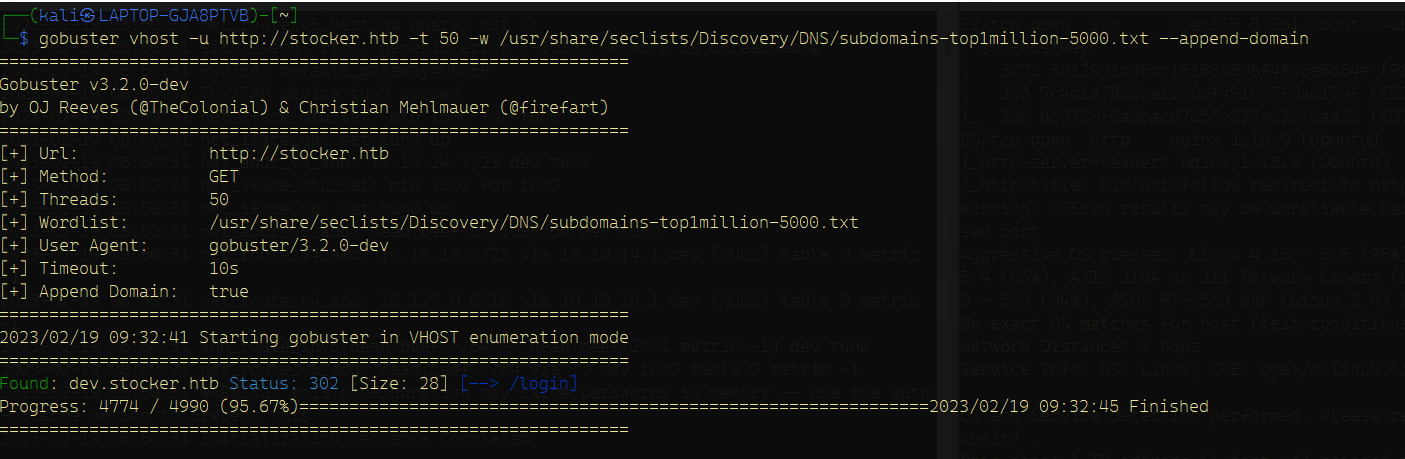
Target pinged

1. Nmap -sS -p- 10.10.11.196
2. Nmap -sS -sV -sC -p22,80 -O 10.10.11.196
3. Put 10.10.11.196 stocker.htb in /etc/hosts
4. Explore website, website source etc
   1. No input field, thus no sql injections nor xss vuln. Webserver version not vulnerable.
5. Discover directories of the website with dirb

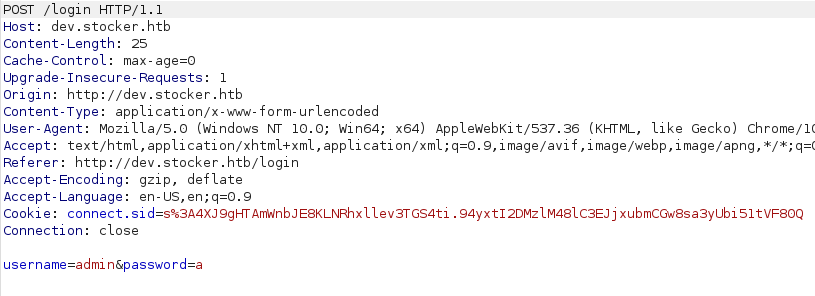
Still nothing interesting

1. Discover subdomains with gobuster



* 1. Finally something! A login page on dev.stocker.htb
  2. First add dev.stocker.htb next to stocker.htb in /etc/hosts

1. Explore login page
   1. Source code == not interesting
   2. Login input not vulnerable to sql injection or xss injection
2. Burpsuite proxy
   1. When trying to login:



* 1. We see that the username and password are passed thru the request
  2. Test for NoSql injection



Payload explanation: first of we need to change the content type to json, because our payload is written in json. Second of all we need to specify the username and password (which are inputs in the login page). In this case they’re both specified as “$ne”: null.

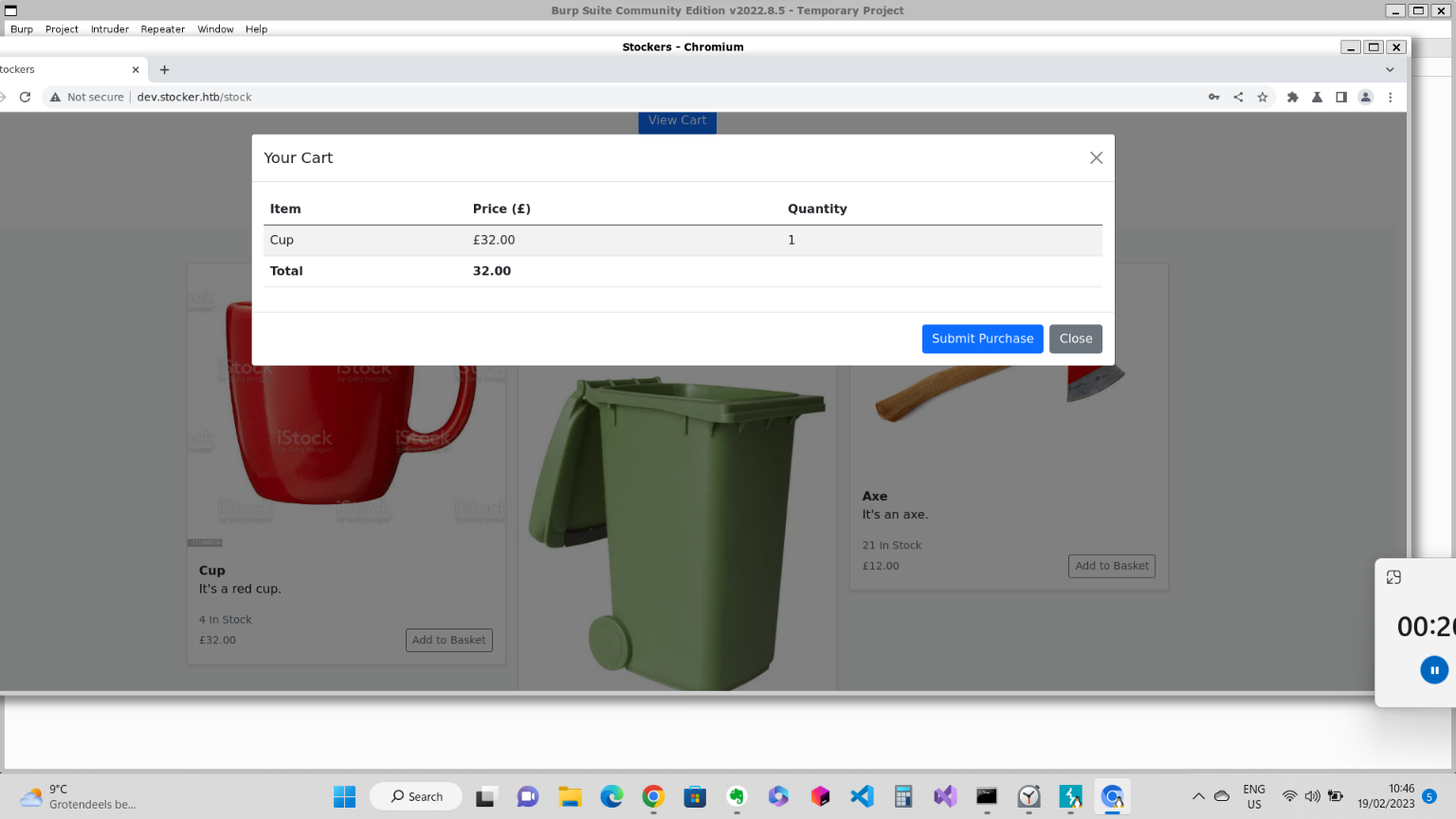
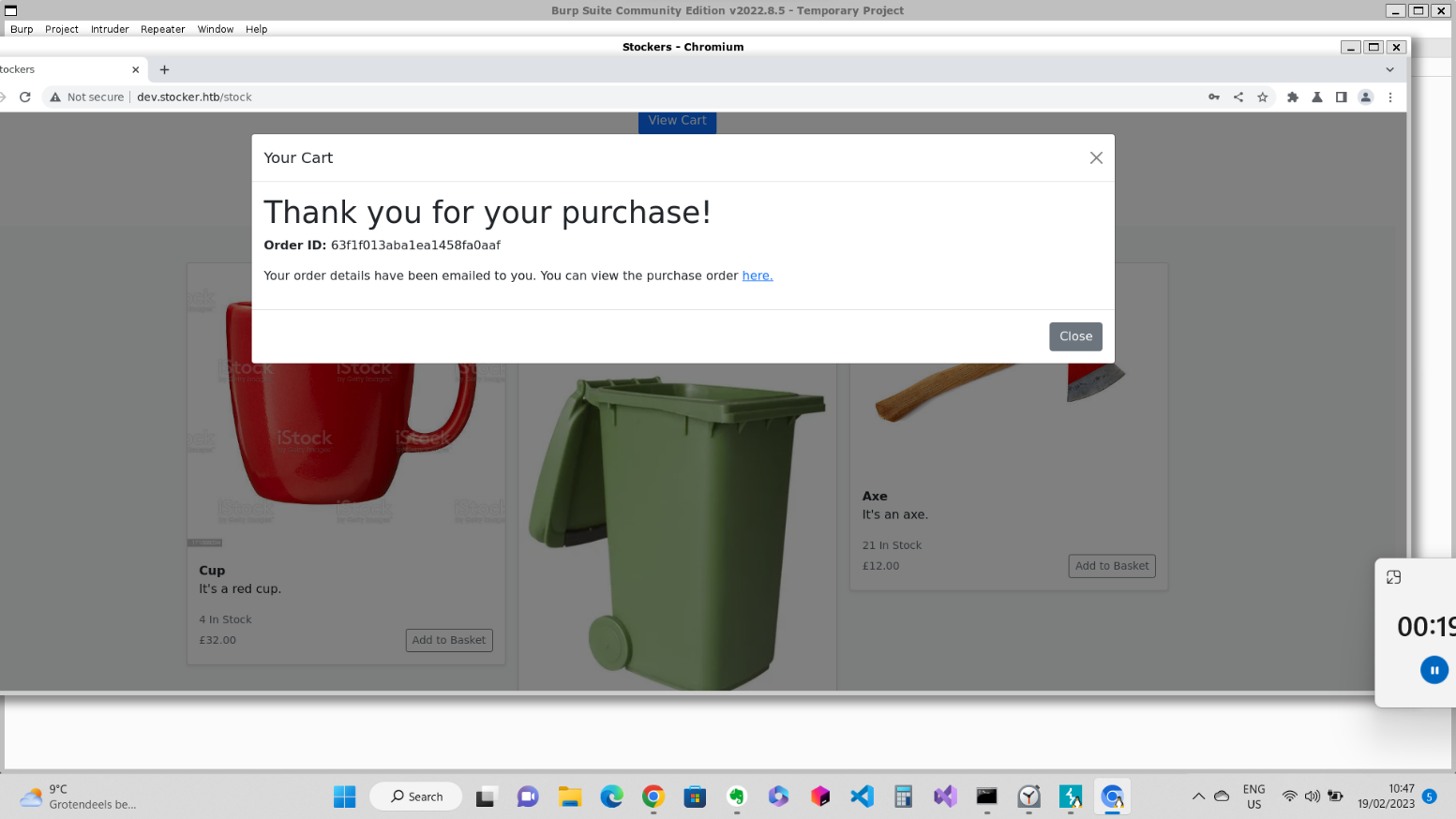
$ne is a special operator used in MongoDB NoSQL queries to match any documents where the specified field is **not** equal to the specified value.

So, in your payload, {"username":{"$ne": null}, "password":{"$ne": null}}, you're essentially asking the MongoDB database to return any documents where the username and password fields are not empty or null.

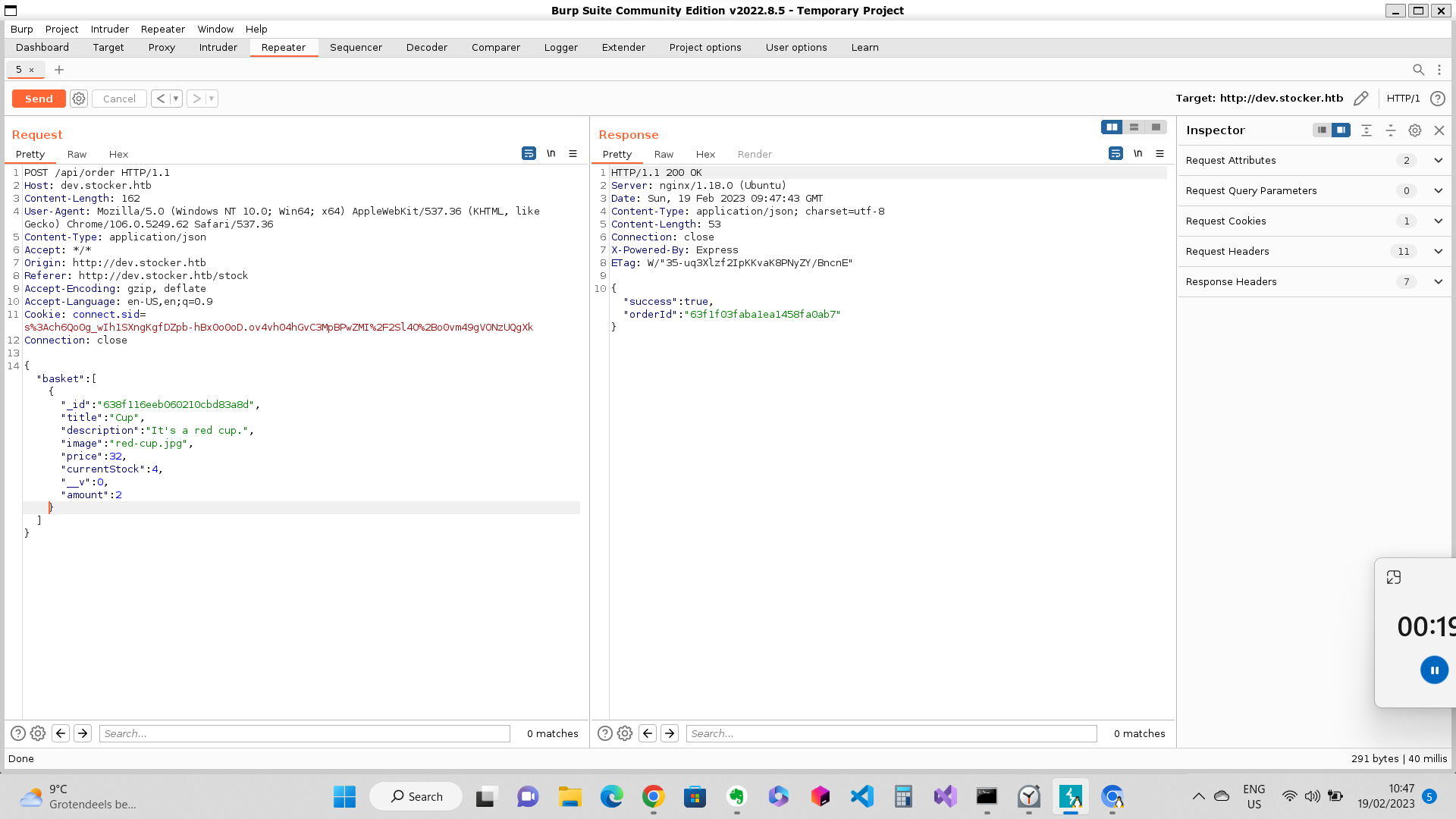
The $ne operator can be used to perform a negative check and can be used with other operators, such as $gt (greater than) and $lt (less than), to create more complex queries.

So basically $ne mean !=

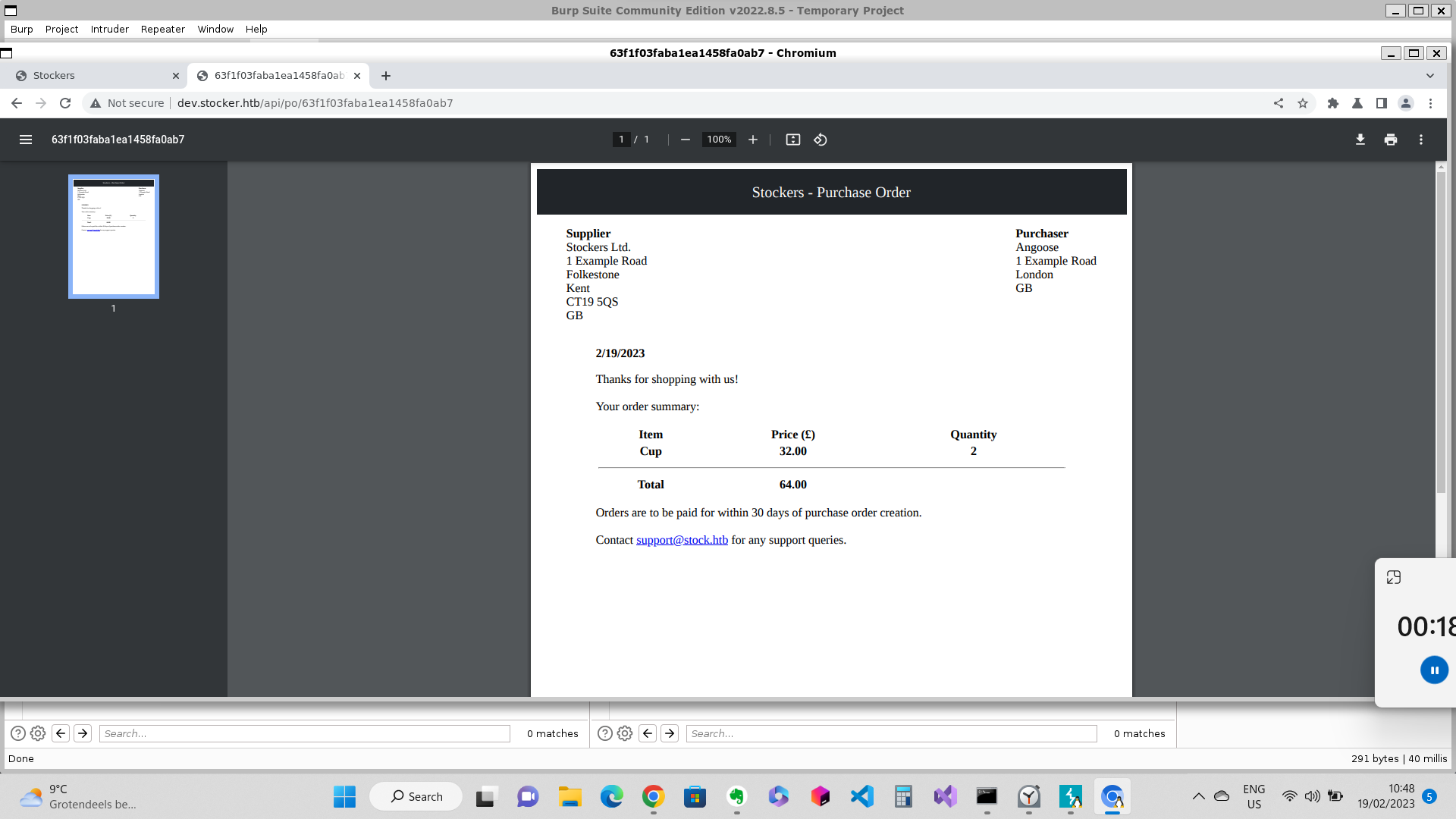
1. The payload works and we’re logged in a page were we can order products. Lets put the cup in the cart and order it:



1. Before clicking submit purchase, we make sure we’re using burpsuite proxy. Now we click and see the request that just have been made

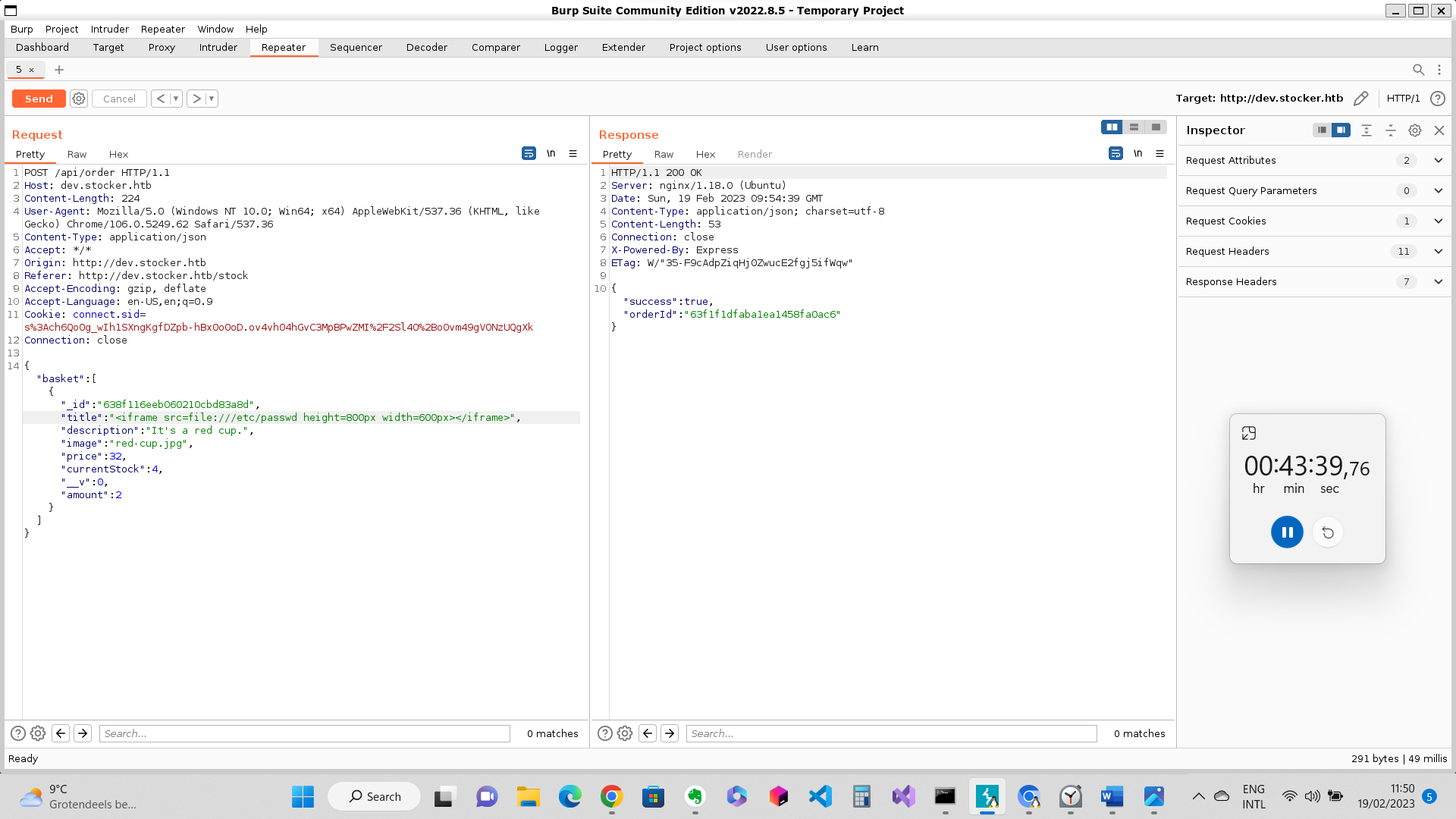


* 1. Send the request to intruder

Here we see our purchase order

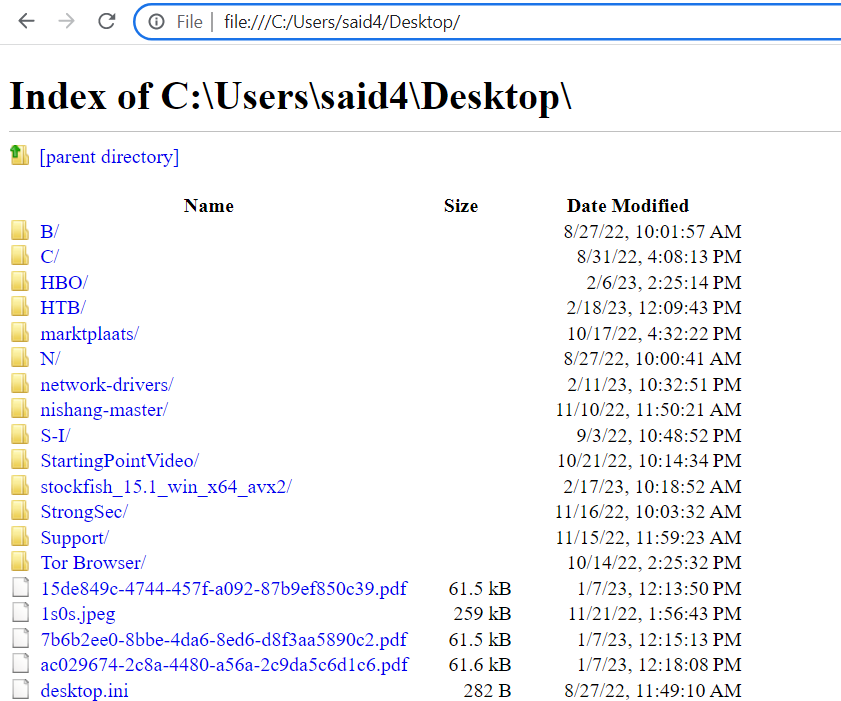
We also see that it’s using an api, which takes as input the order ID.

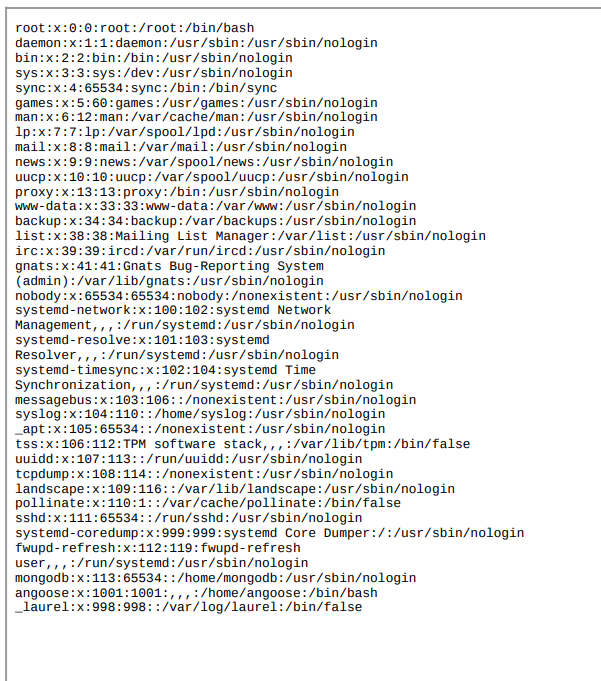
1. Let’s go to intruder, to make a new custom order.



First of some bg info:

The original request is a POST request to the endpoint /api/order on the website dev.stocker.htb. the request has JSON payload in the body that specifies the items to be added to the user’s basket. Normally JSON payloads don’t allow the execution of Javascript or HTML code. However, in some cases, the application may render the JSON data in a way that allows HTML and JavaScript to be executed, which can lead to vulnerabilities like cross-site scripting (XSS). So in this case the web application uses the JSON payload, and pulls out the content of \_id, title, description, image, price and amount. When u see something like this, there’s a chance that the application will not clean or esape the input data in the JSON payload. In these cases the content of let’s say title, will then be interpreted as HTML code. In this case we used the iframe html tag, with the src [file:///etc/passwd](file:///C:\etc\passwd). The src will be interpreted as an url, and since it’s a local file, the content can be displayed.

Here's an example of an url of a local directory:

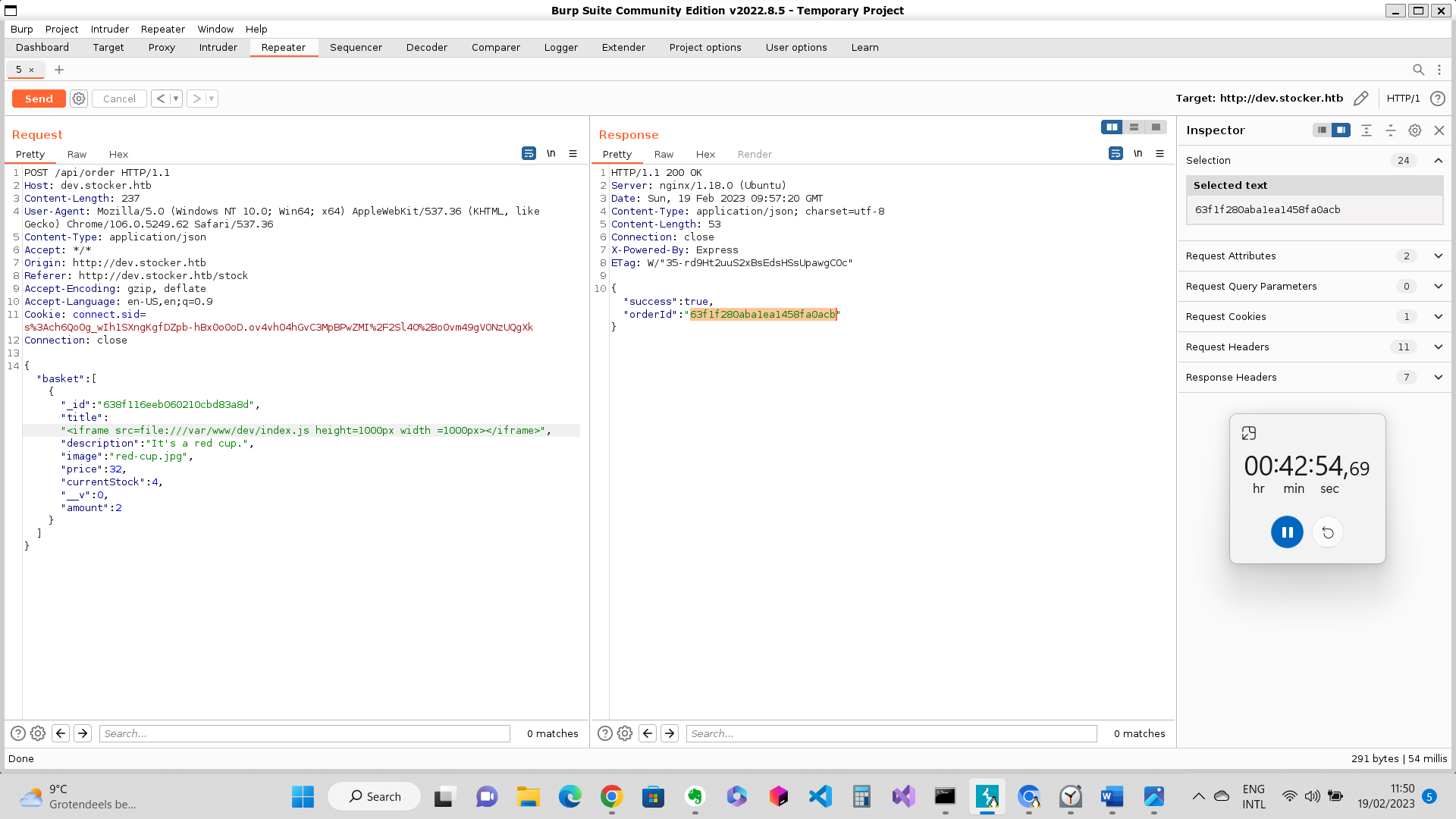
In our case we iframed the content of the passwd file of the machine and this is the result:

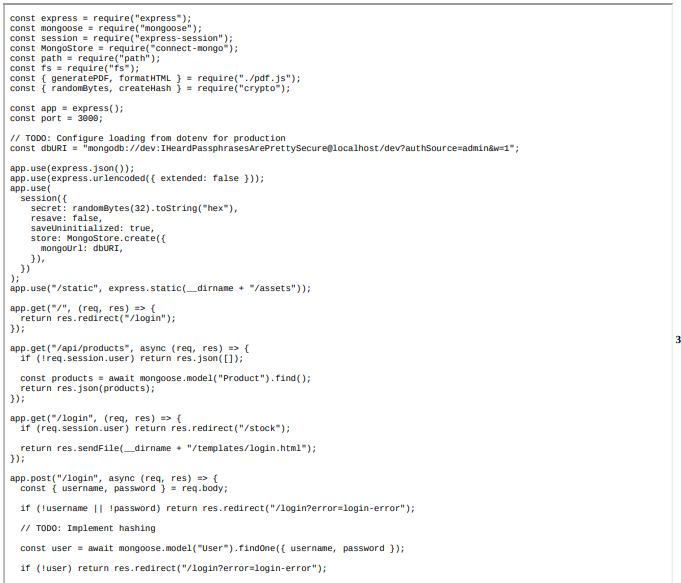
We see here that only to users have access to /bin/bash, which a shell. Since this machines also has port 22 open, it’s pretty abvious what the username for the ssh authentication is: angoose.

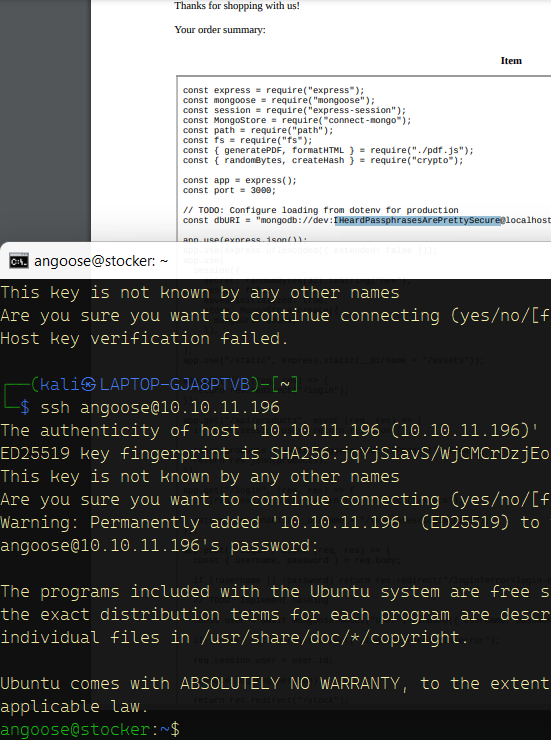
So now we only need to find the password for angoose.

The first file we’re gonna check is located in var/www/dev/index.js, because this file that contains server-side code that interacts with the database and the api. It runs on the server, and is not meant to accessible to clients. If it interacts with the database, the code should somehow have the credentials to do so. If the credentials are put in configuration file, environment variables, or a secure database there’s not much we can do. But if it’s hardcoded in the file, we could just copy it.

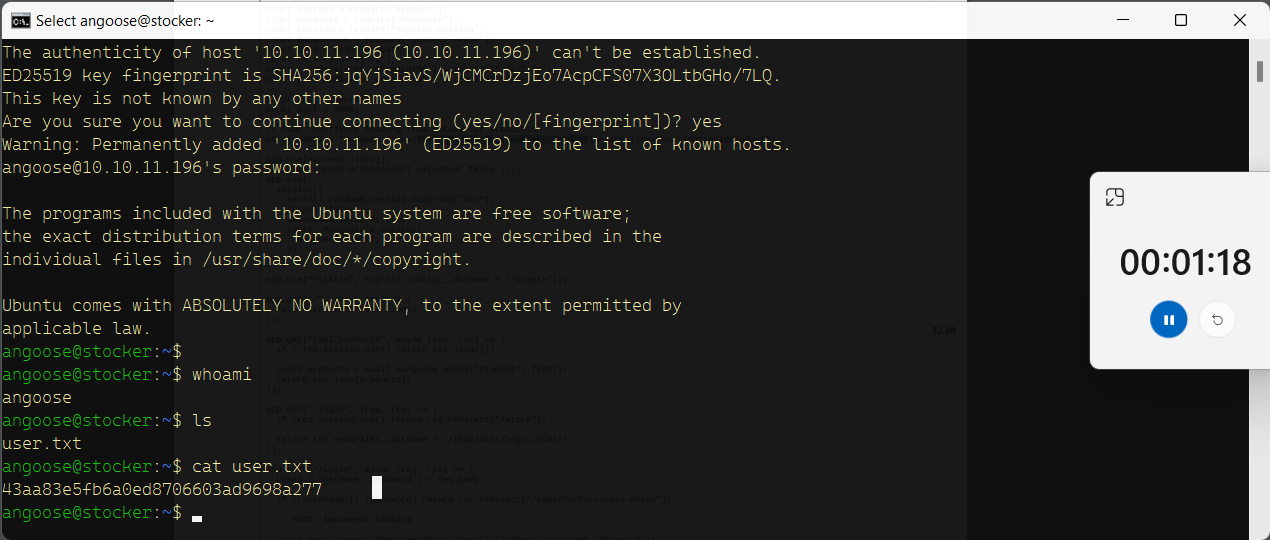
Things we could find in such a file: database credentials and/or API keys

1. Back to intruder, craft html code to display that file

And this what we got back:

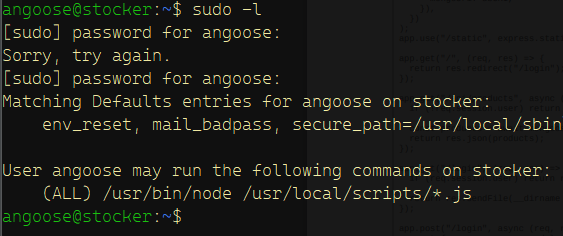
YES, we have a password, let’s hope this pass works for ssh authorization

IT WORKS, WE’RE IN

Let’s get the user flag:

Now the next command we’ll run is the sudo -l command.

The sudo -l command lists the permissions granted to the current user by the sys admin or the sudo config file. It shows the commands and programs that the user is allowed to execute with elevated privileges using the sudo command. This is useful for users who have been granted limited sudo privileges and want to see what commands they are allowed to run as a superuser. Note that to run sudo -l, the user needs to have the appropriate permissions set in the /etc/sudoers file. By default, this command can only be run by a user who is a member of the sudo group on Ubuntu and other Debian-based systems.

After hitting enter we get this:

So this tells us that we’re allowed to run with /usr/bin/node (Node.js executable) javascript scripts located after the /usr/local/scripts/

So lets make a js script that prints out the content of root.txt in the root directory

const fs = require('fs');

fs.readFile('/root/root.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log(data);

});

Const fs = require(‘fs’);

This line declares a constant variable. It is equal to the require function. Require is a built-in function in Node.js that allows u to include external modules. In this case we’re including the fs module. This module provides an API for interacting with the file system, allowing you to read and write files.

Fs.readFile('/root/root.txt', 'utf8', (err, data) => {

This line calles the readFile function of the fs module, which reads the contents of a file. In this case /root/root.txt. further we have to provide the character encoding, in this case utf8. Then we have a third argument, which is a callback function that is called when the file is read. It takes 2 arguments: the err and data. The err will contain an error object if an error occurs. The data will contain the contents of the file as a string.

if (err) throw err;

console.log(data);

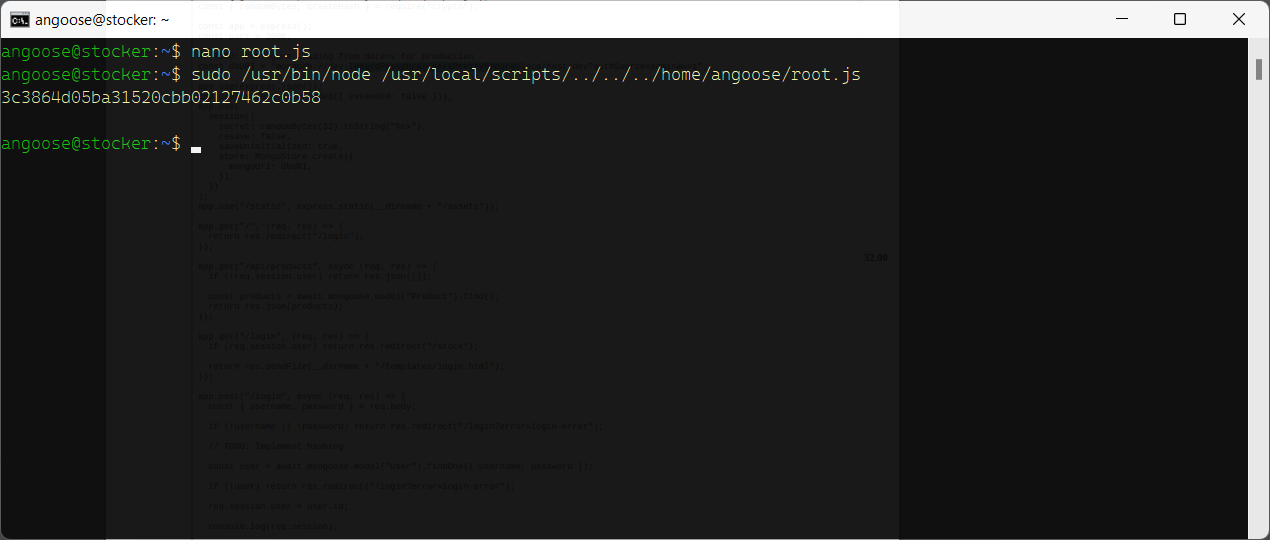
});

This code is inside the callback function passed to ‘readFile’. The first line check if err is NOT equal to null. If err is not null, it means an error occurred while reading the file. In this case the script throws the error by using the throw keyword, which causes the program to stop executing, and prints out the error message to the console.

If there are no errors the first line will be ignored, and we go to the second line. The second line logs the contents of the file to the console.

Now we have to execute the program in the right way. Based on sudo -l, we can run sudo /usr/bin/node /usr/local/scripts/\*.js with root privileges. So we have to execute it this way:

Sudo /usr/bin/node /usr/local/scripts/../../../home/angoose/myscript.js

Aaaand: