Cybersecurity Lab, CSE 3140

Spring 2022

Web Application Security

Section # : 01

Team #: 03

Names: DJ Foley & Jared Moore

Haitham Ghalwash

In a previous project you learned about Web applications that are now part of our everyday activity. In most settings a web application consists of at least four different components. These are the **end use**r, **the browser**, **the web site (hosted on a web server)**, and **usually a database** that stores the state of the system.

Ideally, web applications should be built securely from the ground up – but this is not always the case. The urgency and dependability of web applications in our life, with a product being available, exceeds the rate at which we can ensure a web site is securely designed and implemented. Due to increasing consumer demands, you will find numerous websites going live without ensuring basic security features.

Consequently, various types of existing attacks are targeting web servers, which range from targeting only the web database to large-scale network disruption. Some of the common existing attacks are Cross site scripting (XSS), SQL injection (SQi), Denial-of-service (DoS), Buffer overflow , Cross-site request forgery (CSRF), and Data breaches.

To properly protect web apps from exploitation, we need to ensure some security measures such as date encryption, proper authentication, continuously patching discovered vulnerabilities, and having good software development. The reality is that clever attackers may be able to find vulnerabilities even in a robust security environment. Web application penetration testers make up some of the missed options for securing a web application while building. They are becoming a vital part of the application Security (AppSec) development. You can find bug bounty websites that crowdsource penetration testers to hunt for bugs on major websites.

Learning goal

In this lab we will learn how to exploit vulnerable web applications of the OWASP juice shop project. We will learn how to inject SQL commands, dump database information, bypass authentication, reveal sensitive information and leverage broken access control.

Tools used

Burp Suite, Web Browser, ubuntu VM

Logging onto your VM

We will be doing everything using a remote Linux VM. Each group is assigned a VM using the IP 172.16.50.X (X= 100+ 20*section number + group number) with each VM has two ***usernames***: (***cse1 and cse2). The password is cse3140 for both users***.

172.16.51.59

Option 1: If you are in class, you can simply SSH using MobaXterm into your VM, and use the web browser to access the web interface of juice shop. You will simply type your IP address and use port 3000 to access the juice shop interface.

Option 2: If you are remotely out of class, you will need to SSH with port forwarding to be able to open the juice shop in your local browser

1. **SSH with port forwarding** port forwarding the remote web server running on port 80 to your localhost:3000. You will also need to install burp suite to intercept the tunneled traffic of your browser. Burp suite has an embedded browser that is pre-configured and ready to be used. If you are using Mobaxterm you may find it helpful checking the instructions here.

   Note: *If you choose this option, you need to decide what you want the local port to be (you can choose port 3000 for example). For the rest of the lab, you'll be connecting to your localhost at the set port (localhost:3000). Note on Linux you should have the ability*

to do X Windows forwarding immediately. On Mac you need to install an x server on your machine (such as XQuartz or Mobaxterm). Note if you choose this option, you'll need to install Firefox and an extension

Please ask any teaching member in discord if you are having trouble accessing the environment. There are lots of configurations that can work (and many that won't)! We will be using Burp Suite for most of the lab. Burp Suite comes with the built-in chrome option that you can use to access the juice shop website.

Lab instructions

1- Start Burp Suite, accept all defaults, start a temporary project, and select next until it loads. Your Burp Suite.

2- Connect to your vulnerable web site, using the following link http://IP-address:3000/, the IP address should be the localhost if you are doing the port forwarding using the remote connection over SSH, otherwise the IP-address is simply your VM IP if you are in class.

> **Note**: if you are using tunneling mode you will connect using the specific port chosen previously.

3- Ensure that **Burp Suite** is capturing traffic in the **Site Map** under the **Target** tab. Right click on the URL that is hosting the Juice shop and select **Add to scope**. The URL for you will be http://IP-address:3000. You can use the filter to show only in-scope items in your site Map panel.

Question 1 (5 points)

Walk the "Happy Path" To investigate any web application security vulnerabilities, you need first to walk through the website before you start throwing attack payloads at it. Make sure you fully understand different components and how they work before attempting any exploitation. Walking through the **"happy path" is an effective way** to understand various parts of the application. Just use the application as a normal user to explore different features. In regular software testing this is often called ***"happy path"*** testing.

Before you start your happy path, make sure your burp suite is in the **interception mode**. You are required to browse the site as a regular customer, browse products, make an account, look at different sections, buy some products, etc. The goal here is to interact with the site as much as possible. Burp Suite will be storing all your interactions in the Target window, showing how the site works under the hood. This will make your life a lot easier when trying to figure out how to do exploitation.

Most of what I will be doing involves looking at and making REST API calls. If you have no experience with REST APIs, I suggest looking at this article which should give you an idea of what they do.

Describe three different pages you found. Use the site inspector that is built into most web browsers which can be turned on by pressing F12. What type of web site does this appear to be? Is it strictly html? Are there other technologies being used?

Page 1: The User-Registration Page had me enter an email address, password, repeated password, and a security question before registering. It also showed me the strength of my passwords. There is little to no verification on the registration page so it could be very vulnerable to an attack.

Page 2: The customer feedback page has boxes to enter email address and a comment. It also has a rating bar and a CAPTCHA checker.

Page 3: The Photo Wall has several images posted and a button that allows us to pick an image from our computer and post it to the wall with a caption.

The website appears to use html, javascript, CSS, and SQL based on the code.

Check the products page, what is the administrator's email address? Record any other users' emails you can find.

Administrator Email: bjoern.kimminich@owasp.org

mc.safesearch@juice-sh.op

stan@juice-sh.op

bender@juice-sh.op

uvogin@juice-sh.op

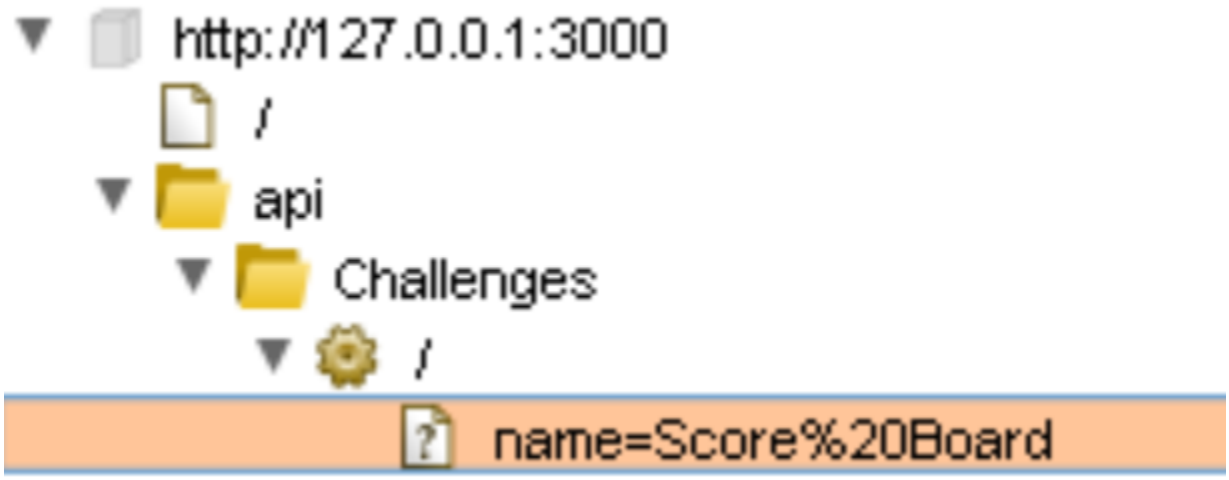
Question 2 (5 points)

Finding the Scoreboard After you have finished walking the ***"Happy Path,"*** your next *goal* is to find the hidden **scoreboard** page. The score board will help you track your progress on the rest of the questions. There are (at least) two ways to go about this question. One is to use the site inspector that is built into most web browsers which can be turned on by pressing F12. The second is to look at Burp Suite for all the requests and responses coming from the web browser. Try to find relevant information that will help you find the hidden scoreboard. Illustrate how you

can find the scoreboard using both methods. Take screenshots and explain the steps you took to find it.

```
▸<a _ngcontent-vmb-c258 mat-list-item routerlink="/score-board" aria-label="Open score-board" class="mat-list-item mat-focus-indicator mat-list-item-avatar mat-list-item-with-avatar ng-star-inserted" href="#/score-board" style>…</a>
  <!---->
  <!---->
```

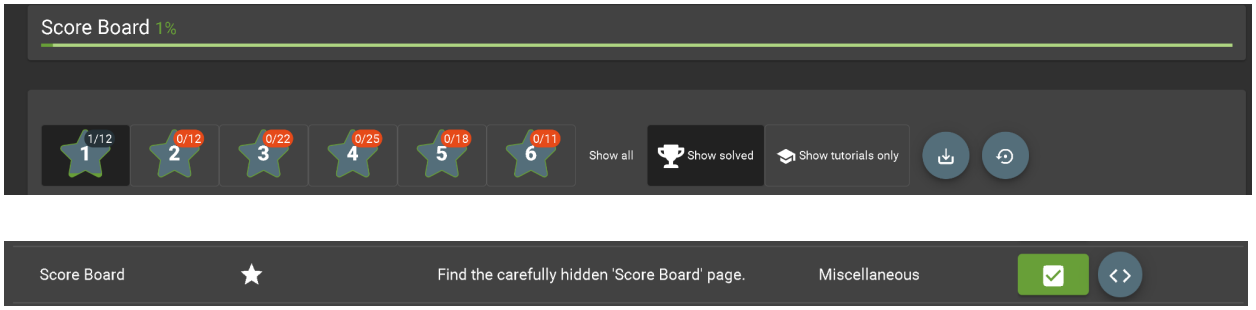
For site inspector, one can simply press F12 and then use Ctrl+F to search for score in the code.




```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Range: items 0-0/1
Content-Type: application/json; charset=utf-8
Content-Length: 623
ETag: W/"26f-07tUVokBjbUExIAfqxIXL4CWjM"
Vary: Accept-Encoding
Date: Mon, 18 Apr 2022 15:01:59 GMT
Connection: close

{
  "status":"success",
  "data":[
    {
      "id":69,
      "key":"scoreBoardChallenge",
      "name":"Score Board",
      "category":"Miscellaneous",
      "tags":"Tutorial,Code Analysis",
      "description":"Find the carefully hidden 'Score Board' page.",
      "difficulty":1,
      "hint":"Try to find a reference or clue behind the scenes. Or simply guess what URL the Score Board might have.",
      "hintUrl":"https://pwning.owasp-juice.shop/part2/score-board.html#find-the-carefully-hidden-score-board-page",
      "mitigationUrl":null,
      "solved":true,
      "disabledEnv":null,
      "tutorialOrder":1,
      "codingChallengeStatus":0,
      "createdAt":"2022-04-11T10:11:05.592Z",
      "updatedAt":"2022-04-17T17:13:38.379Z"
    }
  ]
}
```

For Burp Suite, the Score-Board code can be found under api/Challenges/name=Score%20Board.

Note: provide a screenshot of your scoreboard for each question after you finish the challenge.



Question 3 (10 points)

Access Confidential Documents. Regarding this question, we will try to access the Websites Confidential Documents. This is a known vulnerability in most sites called ***sensitive data exposure***.

Helpful steps:

- Navigate around the website, try finding an FTP file to download. You should come across a file named "**legal.md**".
- Use the "**Target"** tab in Burp Suite to find the URL of the **FTP** server, which stores most of the files used by a business organization. You can even look at the **REST calls** your browser makes to the website by going to **Proxy--> HTTP History** to find out the URL of the FTP file.
- Navigate to the FTP server on Firefox and download the **"secret.md"** file to complete the challenge.

What is the secret token in **"secret.md"**? explain the steps you took to find the FTP server.

Token: dfca3cfddc055396ac65f6bfd73ac396 -

We found the legal.md document in the code and from there navigated to the ftp server with the other documents.

Now try to Download all other files in the ftp server. Which files did you successfully download? Which you did not? What was the error message?

Successful Downloads: incident-support.kdbx, acquisitions.md, legal.md, secret.md, announcement_encrypted.md, order_ed84-64611819f4b5cbd9.pdf

Failed Downloads: coupons_2013.md.bak, package.json.bak, eastere.gg, encrypt.pyc, suspicious_errors.yml

The error message is “Error 403: only .md and .pdf files are allowed!”

Now we learn that Improper handling of error messages can introduce a variety of security problems. Detailed internal error messages to the user (hacker), reveal implementation details that should never be revealed.

To get around this, we will use a character bypass called “Poison Null Byte.” Note that we can download the files from ftp using the url, so we will encode this “Poison Null Byte.” into a url encoded format. The Poison Null Byte will now look like this: %2500. Adding this and then a .md to the end will bypass the 403 error! Null byte injection bypasses application filtering within web applications by adding URL encoded “Null bytes”. Typically, this bypasses basic web application blacklist filters by adding additional null characters that are then allowed or not processed by the backend web application.

Report strings stored in **coupons_2013.md.bak**

n<MibgC7sn

mNYS#gC7sn

o*IVigC7sn

k#pDlgC7sn

o*l]pgC7sn

n(XRvgC7sn

n(XLtgC7sn

k#*AfgC7sn

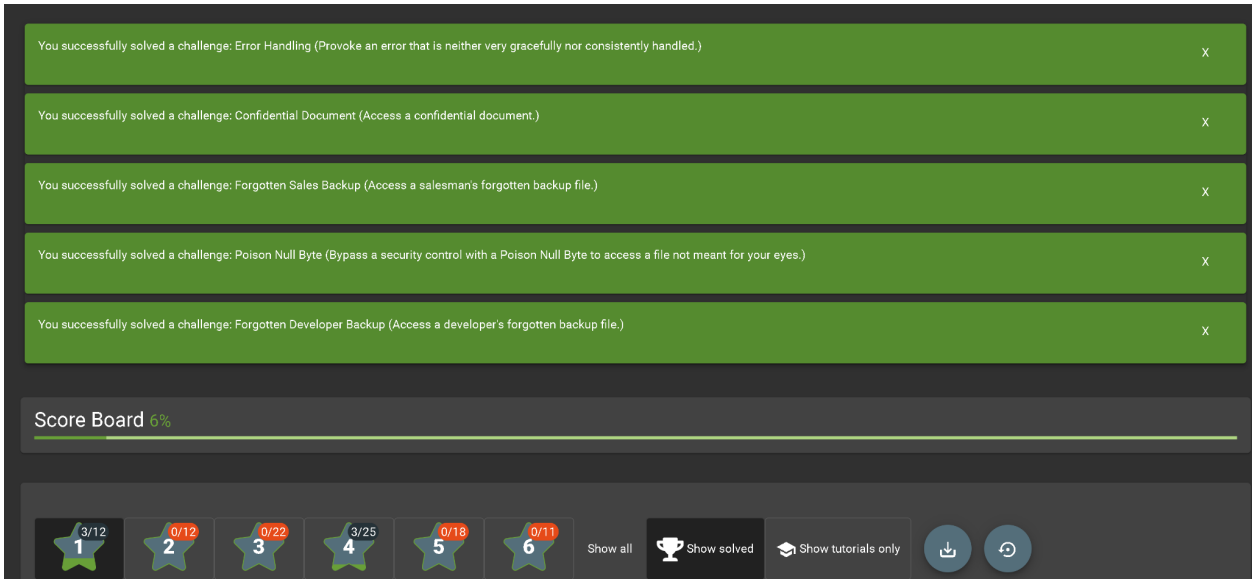
q:<lqgC7sn

pEw8ogC7sn

pes[BgC7sn

l}6D$gC7ss

Note: provide a screenshot of your scoreboard for each question after you finish the challenge.



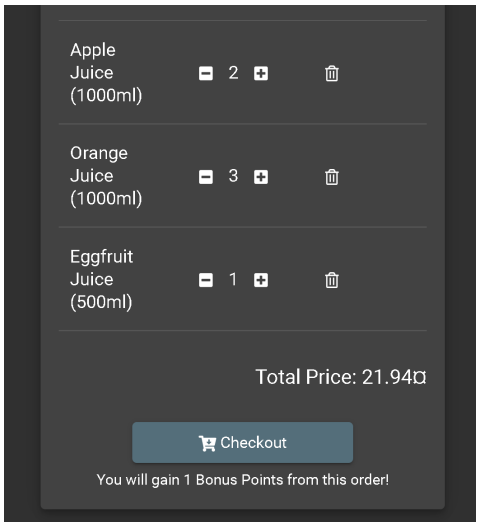
Question 4 (10 points):

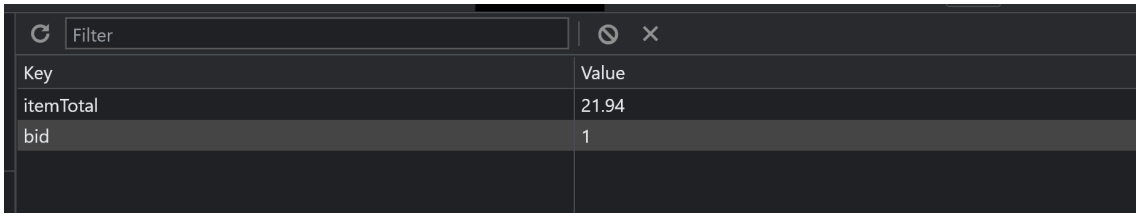
Broken Access Control: Modern systems allow for multiple users to have access to different pages. Different privileges and access levels are usually assigned based on your credentials. Broken Access control exploit can be one of two types: 1) Horizontal Privilege Escalation that allows a user to access other users' data or 2) Vertical Privilege Escalation: when a user can access data or perform an action with a higher level.

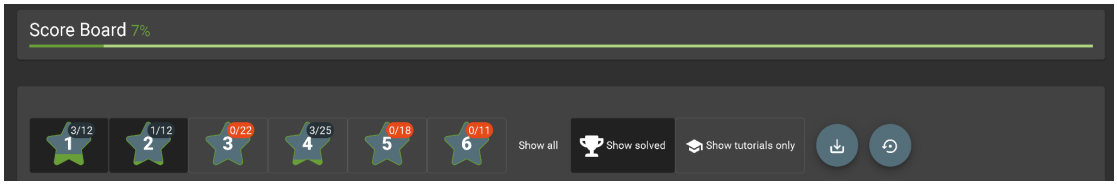
Now, we will try and look at another user's shopping basket. Create an account and add some items to the basket.

Go to Burp Suite and the Target tab. Go to the **"rest"** folder and expand the **basket** folder. You should see an interesting value that is used in requesting your basket. Observe the REST calls your computer is making to the website. And try to view another user's basket using the REST call. You can also use developer tools (inspect element or F12) in your browser to do this.

Take a screenshot of the other users' basket and explain the steps you took to view the other users' basket. How many other users were you able to display?







The checkout and basket page used a value called bid ("Basket ID") to find which basket to select and return the total value of the basket. If you change this bid to be a number lower than the current basket, it will return the page and value for their basket ID. When we did the challenge there were 6 other baskets on the server.

Note: provide a screenshot of your scoreboard for each question after you finish the challenge.

Question 5 (10 points)

Authentication best practices. Describe at least three best practices of authentication on a web site. Provide at least one sentence of why you think this is the best practice or disagree with the recommendation and give rationale for your opinion.

1. Hash all values being stored related to user identity, doing this will separate the domain of one user from other users and make attacks based on proximity of customer data to one another from being a problem.
2. Implement strong password policy requirements, data controls such as salting of passwords and referring to user accounts in a non-identifiable (hashed) way, and making sure the password and customer data that is stored is encrypted and passwords are salted.
3. Use 2 Factor authentication alongside captcha to prevent brute force and phishing style attacks which are not time sensitive. This makes performing attacks require more points of failure for them to be effective and allow a bad actor to take control of a user account.

Question 6 (15 points)

SQL Injection In real web applications, data is usually stored in databases. The web application is constantly constructing the SQL statement to query the database. The constructed SQL statement usually contains data provided by the user, who may be able to inject other SQL statements for the database to execute. This type of vulnerability is referred to as SQL injection, a nice video to watch is here.

In this question, we will try to expose the site's database schema manipulation by using SQL injection. Go to the products page and search for a product that is in the products list like "apple." Now pay attention to the URL which is now updated with the text we just entered. What is the parameter name used for searching?

Now try and enter something completely random that is not in the products list and see if there is any weird behavior. Record your findings.

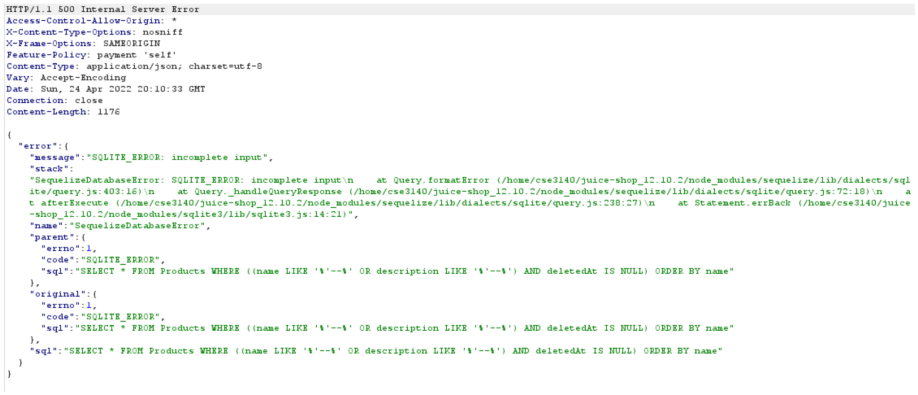
When you enter a product in the search bar that is not in the product list, the search still appears in the url.

You see that your URL in your browser is changing, which indicates that it is making a REST call querying the site's database. This is going to be very useful as we are going to try to exploit the site's underlying database by using these search queries.

Now look at the Search REST calls made by your computer in Burp Suite under **Proxy-->HTTP** History or Look under Target-->"rest" folder-->"search" folder.

Right click on the search for REST call and send it to the Burp Suite Repeater. Go to the Repeater and try searching what you had searched before on the website, adding that instead to the end of the REST call after the "q=".

Observe behavior. **What specific database are they using?**



```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Sun, 24 Apr 2022 20:10:33 GMT
Connection: close
Content-Length: 1176

{
  "error":{
    "message":"SQLITE_ERROR: incomplete input",
    "stack":
    "SequelizeDatabaseError: SQLITE_ERROR: incomplete input\n    at Query.formatError (/home/cse3140/juice-shop_12.10.2/node_modules/sequelize/lib/dialects/sql
    ite/query.js:403:16)\n    at Query._handleQueryResponse (/home/cse3140/juice-shop_12.10.2/node_modules/sequelize/lib/dialects/sqlite/query.js:72:18)\n    a
    t afterExecute (/home/cse3140/juice-shop_12.10.2/node_modules/sequelize/lib/dialects/sqlite/query.js:238:27)\n    at Statement.errBack (/home/cse3140/juice
    -shop_12.10.2/node_modules/sqlite3/lib/sqlite3.js:14:21)",
    "name":"SequelizeDatabaseError",
    "parent":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":"SELECT * FROM Products WHERE ((name LIKE '%'--%' OR description LIKE '%'--%') AND deletedAt IS NULL) ORDER BY name"
    },
    "original":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":"SELECT * FROM Products WHERE ((name LIKE '%'--%' OR description LIKE '%'--%') AND deletedAt IS NULL) ORDER BY name"
    },
    "sql":"SELECT * FROM Products WHERE ((name LIKE '%'--%' OR description LIKE '%'--%') AND deletedAt IS NULL) ORDER BY name"
  }
}
```

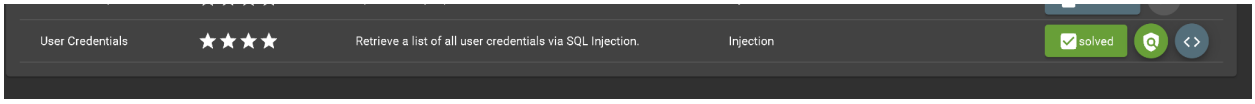
They are using an SQLite3 database.

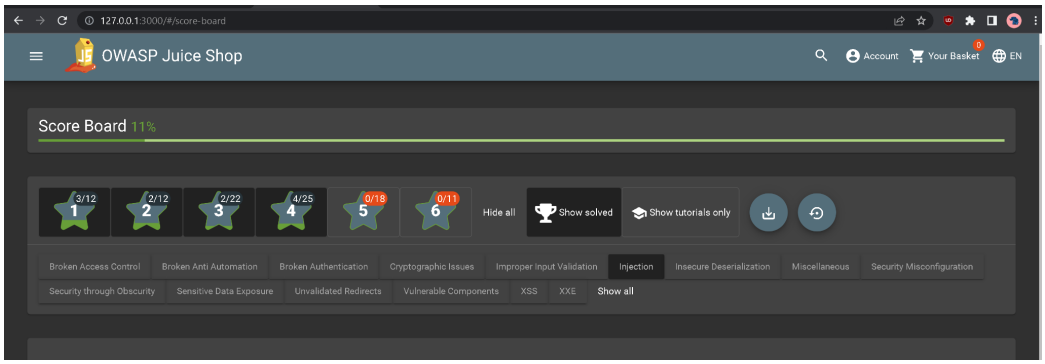
Use these REST calls and queries (entering your queries after the "q=") to **expose the database schema for the site's database**. **How did you expose the schema?** If you haven't had any experience with SQL before I suggest reading a tutorial on it. Once you have the database schema try and expose the ***usernames*** and ***passwords*** of everyone on the site using SQL queries again on the Users table. What type of hash are the passwords in the database using? What is the administrator password? Illustrate your steps with screenshots.

Notes:

- This exercise involves trial and error procedures to get errors and try based on the errors.
- You can submit a request for your rest in the browser by typing the full rest request (ex. http://localhost:3000/rest/products/search?q=apple)
- You can add more SQL queries by appending the appropriate syntax to the rest api POST.

- The UNION SELECT query with the SQLite master table and user's tables is a good start to query extra information. You can try queries by appending the following "')) UNION SELECT 1, 2 FROM table--". at the end of your REST POST request.





To get the information from the table, we had to enter a union sql injection attack with took from the sections throughout the table users. Here, there were 9 values, with the different attributes. Once we know the size of the table, we were able to expose the values stored in this database by combining it with the database we were searching and selecting those values from the database.

Take a screenshot of the database schema and explain the steps you took to expose the database schema.

```
qwert')) UNION SELECT id, email, password, '4', '5', '6', '7', '8', '9' FROM Users--
```

Discus proposed defense mechanisms against SQL injection in terms of network, application and Database.

Input sanitization is the best way of making yourself immune from this type of attack. If the inputs were sanitized, any code written within our query would be treated as text, and this would prevent any of these commands from being run, nullifying our attack.

Question 7 (10 points)

Broken Authentication_Explore the ability to bypass authentication methods. We will rely that most of the web applications user databases rely on having an **admin user.** Developer habits are more common than you expect. Try to enter any username and password and observe the error. What error are you getting? Check whether or not the login form sanitizes SQL before submitting to the server and sending back data. Enter a ‘ in the email field and any password. What error are you getting? What do you think is happening?

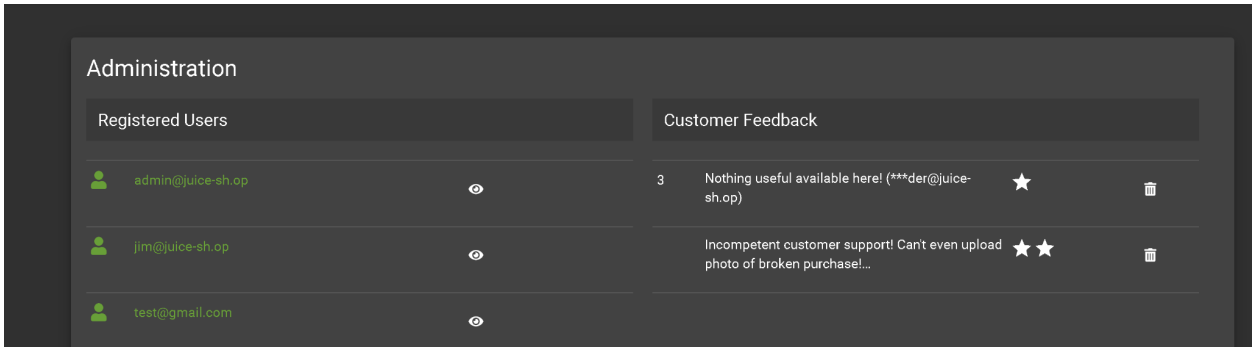
When you enter an incorrect email and password the error states “Invalid email or password.” When you enter a single quote as the username the error states “[object Object]”. When you enter any password in the password field, the computer checks the password and then sends back a request saying that this failed. What we want to do is make sure this password check never runs, so we will use `-- which will cause the server to never get to or run the password checking code.

(Part 1) You probably understand how SQL injection can be used to do different things. The login form is usually a SQL query that is checking the username and password fields provided. Try to bypass the password login checking by adding a **“closing quotation followed OR operator and a true condition (1==1)”** in the login field with any password. Remember to comment on the rest of the sql statement using(--). What do you think just happened? What e-mail address are you logging with? Try logging using that email address and append '-- at the end, use any password. Were you able to login?

The email address I am logged in as is the admin, admin@juice-sh.op. Appending it with “--” does not login. When the server checks to see if the password I use is correct, it tests a condition to check if it is true. By putting the OR 1==1, its essentially saying “let the user enter the page is the password == stored password or 1==1” and since 1==1 is true all the time, no matter what is in the password field it will treat it as if the whole condition was true.

(Part 2) create an admin account on the website that will give you complete control of the website. First logout and create a new account with an email and password. Using the Burp Suite, observe the REST calls you are making and responses from the website. Is there something in those REST calls that you can use to create your own admin account?

Log in to your newly created admin account and navigate to http://localhost:3000/#/administration to open the admin section of the website. Delete all the good reviews on the website.



Take a screenshot of what you did to create an admin account, a screenshot of the admin page and explain the steps you took to get there.

Hint: Use the Repeater tab in Burp Suite to make your own REST calls to the website.

Question 8 (15 points)

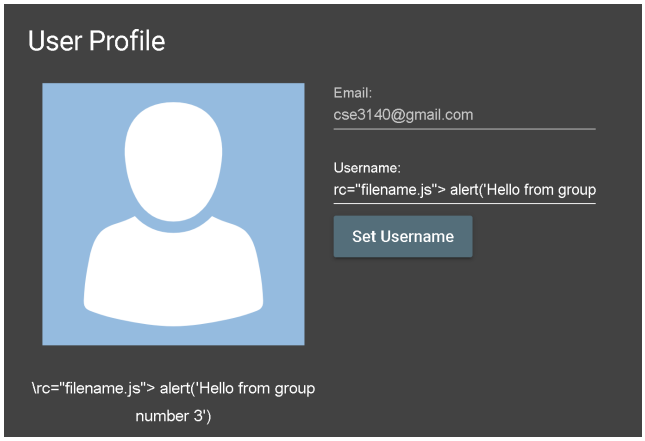
JavaScript programs can be inserted anywhere into an HTML document using the <script> tag. In this question we will work on learning basics about JavaScript that is widely used for web attacks.

<html>

<body>

```
  <p>Before the script...</p>

  <script> alert ('Hello, world!'); </script>

  <p>...After the script. </p>

</body>

</html>
```

add the parameter **src =”filename.js”** to your script tag. Let us include a different message “Hello from group number X.” Which message was displayed? Why?



The message that was displayed was “\rc=”filename.js”> alert(‘Hello from group number 3’)” because it was sanitized.

Now let’s impede the source code in line with the **src** parameter, this is usually how XSS attacks are carried out. Let us start by examining a DOM XSS. Usually, for unsanitized inputs, we can inject some elements to the HTML page. Start by placing the following HTML tag in the search box.

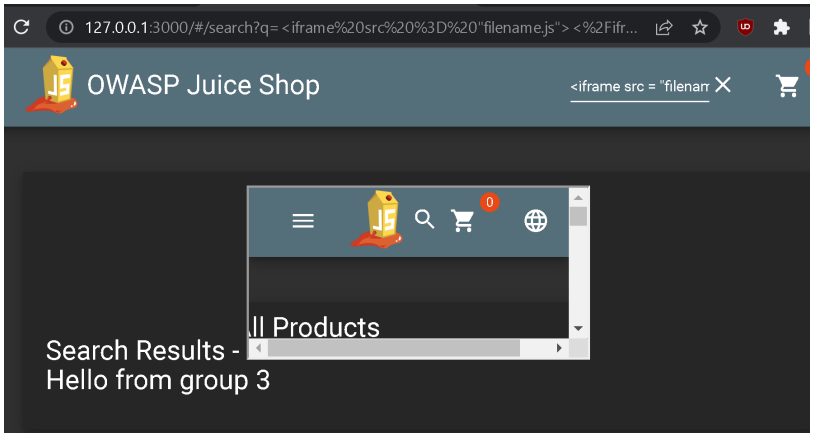
```
<h1>hello world</h1>
```

What did you notice? Explain.

The search results came up as just “hello world” without the tags. This is because the tags are interpreted as html language syntax.

Another widely used tag for web attacks is the inline frame. Inline frame is just one "box" and you can place it anywhere on your site. Frames are a bunch of 'boxes' put together to make one site with many pages. So **<iframe>** just brings some other source's document to a web page. The <iframe> are used to specify inline frames or floating frames. Let us

use iframe to execute a basic java script from question 8. Inject your “Hello from group X” from the previous question into the search box. Describe your attack. What do you notice?



The iframe creates a smaller window in the search results and the text is labeled next to it. The iframe’s source is the page itself filename.js, meaning it shows a mini copy of the current page in the iframe.

Let's now perform a persistent XSS, First login to the admin account, then navigate to the last Login IP page. We will add a header field named True-client-IP in the header request

A Cross-site scripting (XSS) is a type of code injection attack, which typically involves three entries. An attacker, a victim, and a target website. Getting a piece of code into the victim’s browser can be easy. Anytime the victim browser visits the attacker webpage, the code will run in the victim’s browser. However, due to sandbox protection the code from the attacker will not be able to affect the page from the target website. To affect the interaction with the target website, the code needs to come from the target website. The attacker needs to inject the malicious code into the victim's browser via the target website. This term “cross-site” is used also in Cross-site Request Forgery (CSRF) attack, which is different from (XSS). What is the difference between (XSS) and (CSRF)? What damage can XSS cause? **There are two known types of XSS, Non-persistent XSS and persistent XSS.**

One of the ways you could have someone else's browser render a piece of code is with cross site scripting, where the browser of the website user renders the malicious code, however this doesn’t run anything malicious on the server side. Cross site request forgery uses someone's credentials or this cross site scripting to perform actions on their behalf they did not intend, such as masking a site to make the user think different applications do different things (such as changing the delete my account button into “login now”) or directly using the users credentials/changing the requests they make before they are sent to the web server. Damage of XSS can cause downtime and inaccessibility for users, and also can be hidden in things like usernames and passwords, which may not be run directly or on every usecase, making these types of attacks ticking time bombs waiting for the wrong screen to be rendered.

Try to test at least two types of XSS examples. document your steps and findings with screenshots. How can you prevent XSS? What is the difference between XSS and SQL injection?

XSS can be prevented by sanitizing inputs properly to ensure all code-reminiscent inputs are replaced with visually equivalent but non-readable machine code characters, and ensuring all user inputs and forms are sanitized by the server before being able to be posted up. If the users are able to reach each other directly, or post up comments and other bits of code inside web pages that can be automatically rendered, that can lead to problems. The Difference between XSS and SQL injection is XSS places front end code like javascript and html in places in which the code may be rendered for users, whereas SQL injection manipulates the SQL queries which hit the server side databases, and either gives them poisonous commands to drop tables, and retrieve user information, or prevent the code from running important steps that happen later in the line, such as password verification.

Question 10 (10 points)

List at least two other web security problems that you have not discussed in this lab. In addition to listing the problem, describe at least one recommendation for dealing with the problem.

1. You can send multiple requests to the webserver within ten seconds using a singular CAPTCHA ID. This means that a user could theoretically spam the main server manageably, and the CAPTCHA ID's are not individualized by request or made unusable after being used for a single request. This is a vulnerability which could, in conjunction with a ddos attack, bring down the webserver. To deal with the problem, you should have a captcha work only once per generation.
2. The web server also had an issue with the storage of user data and passwords. None of the passwords were stored securely in SQLite, and this meant that once I was able to have the server output the information, it was immediately accessible and usable.

Question 11 (10 points):

In most modern web applications you will be serving ads from some external provider. Assume that your ad partner is not willing to have you host and control their ads. Explain how you would serve these ads in the most secure manner so that the advertiser cannot arbitrarily interact with the rest of your code.

A distributed content delivery network for ads would allow for faster delivery directly to customers, and you could also require a local cache to serve the ads directly. Another way advertisers are choosing to serve ads which is less detectable is native content ads. Ad blockers work by detecting the domain and IP address of where certain types of content comes from, rerouting the DNS to a list of blacklisted advertising domains to localhost to make those resource requests fail. If the content is sent to an intermediary first, and then the CDS is used as a proxy for the ads, they will look as if they are coming from the serving website, without extra cost to the content provider. This has the benefit of routing all traffic through the CDS to decrease load directly on the content provider and allow for fast rendering and caching close to content consumers.