# **THM – OWASP-Juice-Shop**

## Note

* This THM room is guided. The intention of this room was to provide the basic understanding of how to perform Cross-Site Scripting (XSS) attacks as well as showcases why they work.

## Task 1

* N/A

## Task 2

### Question #1 – What’s the Administrator’s email address?

* Browsing around each product, we identify three available email addresses from some reviewers:

1. [admin@juice-sh.op](mailto:admin@juice-sh.op)
2. [bender@juice-sh.op](mailto:bender@juice-sh.op)
3. [jim@juice-sh.op](mailto:jim@juice-sh.op)
4. [uvogin@juice-sh.op](mailto:uvogin@juice-sh.op)

* Evidently, the Web Admin had accidentally left a review on their own website with a privileged account.

### Question #2 – What parameter is used for searching?

* The search icon on the top of the page reveals an input field that we can use to search for something. Upon entering an arbitrary value, we can take note of the query parameter that is provided in the URL. The resultant query parameter used here is: ?q

### Question #3 – What show does Jim reference in his review?

* Doing a text-to-Google search reveals that “Fresh out of a replicator” is a likely reference to Star Trek, Replicator.

## Task 3

### Question #1 – Log into the administrator account

* Navigating to the login page, we are given a form with two inputs. Using Burp Suite to intercept a POST request to our local machine, we can use the Repeater tool for some injection attacks.
* The simplest injection attack we can perform with little effort would be a simple SQL Injection attack, using the **‘ or 1=1--**command.
* The initial apostrophe ‘ closes out the current SQL statement that is used in the backend code to evaluate the user input.
* The OR condition entails a “one or the other” condition.
* The 1=1 statement is a simple condition requirement. Being that 1 is indeed equal to 1, it will always be true.
* The last double dashes -- are to comment out the remaining SQL code in the backend.
* As a result of this SQL Injection attack, we log into the site using the first available account – which happens to be the Administrator account.

### Question #2 – Log into the Bender account

* Given that we were able to identify a total of four email accounts, we can attempt to log into another user’s account by simply replacing the **‘ or 1=1** with the email account of interest. It’s important to note that the first single apostrophe we use is appended at the end of the email address, as we want to include the email address in this case. Thus, the final result will appear as: **{email\_address}’--**
* It’s also important to note that the reason we can supply an arbitrary password is likely due to the password variable being commented out – hence why supplying only the email address is all that is needed for this to work.

## Task 4

### Question #1 – Bruteforce the Administrator account’s password

* While we have successfully logged into the Administrator account without supplying the correct password, we should ideally try and identify the correct password. One method is to use a Brute-force attack. We can do this easily – albeit with some throttling limits being set in place – with either Burp Suite. For this question, the “best1050.txt” wordlist was recommended.
* Note: It’s important to note that selecting the appropriate wordlist determines whether the brute-force attack is successful or not. Simply using a popular wordlist does not guarantee if credentials are going to be revealed.

### Question #2 – Reset Jim’s password

* Sometimes, you may encounter a website that has improperly setup their “Forgot Password” feature on their login page. This entirely depends on what information is made available, such as if the presence of Security Questions appear, what other information was found that could be used to exploit the Forgot Password feature.

## Task 5

### Question #1 – Access the confidential document

* Post login-exploitation procedures should involve exploring the previously restricted endpoints to both: verify if the user has authorisation, and explore what protected information is available – such as technical documents, HR documents, financial documents, etc. The ultimate goal being to map out the website from top to bottom.
* Sensitive Data Exposure also looks at examining the authorisation policies that have been instilled on the website as well, as it is possible to come across a website that has either neglected or improperly setup authorisation or requirement policies for their user account(s).
* Methods to explore for SDE should involve LFI or File Inclusion practices.

### Question #2 – Log into the MC SafeSearch account

* Sometimes, when attempting to brute-force an account, L33tsp34k should be worth considering as part of password cracking rules. There is nothing more to this question as the information is made available to us in the question.

### Question #3 – Download the Backup file

* In the event that you do get access to the file repository on a webserver, chances are that some files and folders are still restricted due to the user account that’s been used to host the webserver. Realistically, the account that is used for hosting the webserver is given “least privileged access”, which means that access to sudo will require different credentials.
* Alternatively, you can use existing exploits, such as a **“Null Byte Injection”** to get around protected files. This exploit appends the **%00** to whatever URL or string is supplied. Null Byte Injection only works if the website does not sanitise or correctly handle such characters.
* Null Bytes are used to terminate strings or be a placeholder in code. Because of the use case of Null Bytes, we can append these string of characters into a URL to cause the site to incorrectly handle the modified endpoint and thus return protected content. This is commonly used in LFI/File Inclusion attacks.

## Task 6

### Question #1 – Access the administration page

* There are two types of Broken Access Control exploits: **Horizontal** and **Vertical Privilege Escalation**.
* Horizontal PE = when a user can perform an action or access data of another user with the **same level of permissions**.
* Vertical PE = when a user can perform an action or access data of another user with **higher level of permissions**.
* Note that this is only applicable on sites that either: have incorporated their CMS setup, or are using an existing CMS solution, such as Shopify, Wix, or Wordpress, etc.
* Each site will implement their own Administration panel differently. If it’s an existing CMS solution, it is typically pre-defined with its own endpoint. However, if it’s a custom Admin panel, you will have to either enumerate the file paths, or explore the available code in the debugging menu (if the site has been built with a JavaScript framework, the Routing may have already been specified).
* Looking at the available main-es2015.js file, we can perform a quick search for anything “admin” related. Looking through each search result, we are presented with **“path: administration”**. If we visit /#/administration, it appears to be an active endpoint, but there is some backend logic involved to prevent standard users from accessing the endpoint. We could try accessing the location using the admin account we brute-forced in the previous question. Upon re-visiting the endpoint, we are given the flag.

### Question #2 – View another user’s shopping basket

* One of the important aspects to consider when working with an e-commerce site is protecting a user’s shopping cart. Failing to protect unauthorised snoopers from viewing other user’s carts may not sound impactful, but such an exploit could potentially lead to credit card snooping, or extracting sensitive information that no other user should have access to, such as address details. This, of course, will vary from website to website depending on what information is stored in their database.
* To get this flag, we can simply use Burp Suite’s Repeater tool and capture the inbound HTTP requests we receive. Careful examination shows that the /rest/basket/ endpoint is used to store the shopping carts for other users. If we append a random number at the end, such as 1 or 2, we can quickly see that shopping carts on this website are not protected at all, and a simple script kiddie can view the details of shopping carts through simple numerical increments of 1. Forwarding the modified endpoint to /rest/basket/2 reveals the flag.

### Question #3 – Remove all 5-star reviews

* An interesting point to explain is the ability to remove reviews through the web admin account. This could be impactful as an attacker might be able to automate a script and remove any and all product reviews.
* The action of removing the 5-star review reveals the flag.

## Task 7

### Question #1 – Perform a DOM XSS

* Remember, a DOM based XSS attack involves injecting JavaScript inside HTML elements that’s rendered on a page in response to a request, such as missing an input field in a form submission.
* There are 4 primary types of XSS attacks: DOM, Persistent, Reflected, and Blind.
* To test for a DOM XSS vulnerability, we can look at injecting either: an <iframe /> or inline <script></script> code.
* Note: this only works if the site’s input fields are not filtered or sanitised, as without the sanitation, the site will send a new HTTP request and re-load itself with the newly added JavaScript injection.
* Using the <iframe /> technique, we are presented with the flag for this task.

### Question #2 – Perform a Persistent XSS

* Remember, a Persistent XSS attack looks at inserting JavaScript malicious code into a database, so that any subsequent HTTP request made to the page (that calls the database) will be reflected to the subsequent user.
* Example: if Johnny successfully pulls off a Persistent XSS attack, then Sarah from your next-door neighbourhood will come across the same page, and will unknowingly execute the JavaScript code on her end.
* To get this flag, login using the admin account and visit the Account > Last Login IP page. The value of the existing IP does not matter here. Open Burp Suite and ensure it’s been setup to intercept requests on the target domain. Once listening, logout of the admin account. In the intercepted HTTP request, add an additional header parameter of: “True-Client-IP” followed by the value “<iframe src=”javascript:alert(‘persistent xss’)”>”. Once added, forward all requests until no more requests are captured. Turn off Burp Suite. Revisit the same page but without Burp Suite on. Upon doing so, we get the flag.
* *True-Client-IP* is a property in the Header that captures the user’s public IP address.
* This only works because there is no server-side validation when capturing the user’s IP address. So an attacker could easily substitute the value for *True-Client-IP* with malicious JavaScript.

### Question #3 – Perform a Reflected XSS

* Remember, a Reflected XSS attack is one in which the exploit is immediately returned to the user on view.
* To get this flag, login as the administrator and navigate to the Order History page. If we explore an order, we are given the ID of the order in the URL and on screen. If we try using the same <iframe /> technique and refresh the page, we quickly identify that a Reflected XSS attack is present on the site. Thus, we get the flag.