**OWASP Juice Shop**

Chúng ta sẽ cùng xem xét [OWASP's TOP 10 vulnerabilities](https://owasp.org/www-project-top-ten/) trong 1 web cụ thể **Juice Shop**

[Injection](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection)

[Broken Authentication](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication)

[Sensitive Data Exposure](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure)

[Broken Access Control](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control)

[Cross-Site Scripting XSS](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS))

1. Inject the juice

|  |  |
| --- | --- |
| **SQL Injection** | SQL Injection là khi kẻ tấn công nhập một truy vấn độc hại hoặc không đúng định dạng để truy xuất hoặc giả mạo dữ liệu từ cơ sở dữ liệu. Và trong một số trường hợp, có thể đăng nhập vào tài khoản. |
| **Command Injection** | Command Injection là khi các ứng dụng web lấy dữ liệu đầu vào hoặc dữ liệu do người dùng kiểm soát và chạy chúng dưới dạng lệnh hệ thống. Kẻ tấn công có thể can thiệp vào dữ liệu này để thực hiện lệnh hệ thống của riêng chúng. Điều này có thể thấy trong các ứng dụng thực hiện kiểm tra ping không đúng cấu hình. |
| **Email Injection** | Email Injection là một lỗ hổng bảo mật cho phép người dùng độc hại gửi tin nhắn email mà không được máy chủ email cho phép trước. Điều này xảy ra khi kẻ tấn công thêm dữ liệu bổ sung vào các trường, không được máy chủ diễn giải chính xác. |

**Câu 1: đăng nhập với tư cách quản trị viên**

Sau khi điều hướng đến trang đăng nhập, hãy nhập một số dữ liệu vào trường email và mật khẩu.



**Trước khi** nhấp vào gửi, hãy đảm bảo  **chế độ Chặn được** **bật** .

Điều này sẽ cho phép chúng ta xem dữ liệu đã được gửi tới máy chủ!

A screenshot of a computer

Description automatically generated

Bây giờ chúng ta sẽ đổi ký tự " **a** " bên cạnh email thành:  **' or 1=1--** và chuyển tiếp email đó đến máy chủ.



**Tại sao điều này lại hiệu quả?**

1. Ký tự **'** sẽ đóng dấu ngoặc trong truy vấn SQL
2. ' **OR** ' trong câu lệnh SQL sẽ trả về true nếu một trong hai vế của nó là true. Vì  **1=1 luôn đúng** , nên toàn bộ câu lệnh là true. Do đó, nó sẽ cho máy chủ biết rằng email hợp lệ và đăng nhập chúng ta vào  **id người dùng 0** , tình cờ là tài khoản quản trị viên.
3. Ký  tự **--**  được sử dụng trong SQL để chú thích dữ liệu, bất kỳ hạn chế nào đối với việc đăng nhập sẽ không còn hiệu lực vì chúng được hiểu là một chú thích. Điều này tương tự như  chú thích **#** và  **//** trong python và javascript.

A screenshot of a computer

Description automatically generated

**Câu 2: đăng nhập với tư cách Bender account**

Tương tự như những gì chúng ta đã làm trong  **Câu hỏi số 1** , bây giờ chúng ta sẽ đăng nhập vào tài khoản của Bender! Ghi lại yêu cầu đăng nhập một lần nữa, nhưng lần này chúng ta sẽ đặt:  **bender@juice-sh.op'--** làm email.



Bây giờ, chuyển tiếp nó tới máy chủ!

Nhưng tại sao chúng ta không đặt  **1=1** ?

Vâng, vì địa chỉ email hợp lệ (sẽ trả về  **true** ), chúng ta không cần phải buộc nó phải là  **true** . Do đó, chúng ta có thể sử dụng  **'--** để bỏ qua hệ thống đăng nhập. Lưu ý rằng **1=1** có thể được sử dụng khi email hoặc tên người dùng không được biết hoặc không hợp lệ.

A screenshot of a menu

Description automatically generated

1. **Who broke my lock?! – Broken Authentication**

Weak passwords in high privileged accounts

Forgotten password pages

A screenshot of a computer

Description automatically generated

In this task, we will look at exploiting authentication through different flaws. When talking about flaws within authentication, we include mechanisms that are vulnerable to manipulation. These mechanisms, listed below, are what we will be exploiting.

Weak passwords in high privileged accounts

Forgotten password pages

 More information: [Broken Authentication](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication)

**Question #1: Bruteforce the Administrator account's password!**

We have used SQL Injection to log into the Administrator account but we still don't know the password. Let's try a brute-force attack! We will once again capture a login request, but instead of sending it through the proxy, we will send it to Intruder.

Go to Positions and then select the **Clear §**button. In the password field place two § inside the quotes. To clarify, the § § is not two sperate inputs but rather Burp's implementation of quotations e.g. "". The request should look like the image below.

A screenshot of a computer

Description automatically generated

For the payload, we will be using the **best1050.txt from Seclists**. (Which can be installed via: **apt-get install seclists**)

*You can load the list from: /usr/share/wordlists/SecLists/Passwords/Common-Credentials/best1050.txt*

Once the file is loaded into Burp, start the attack. You will want to filter for the request by status.

A **failed**request will receive a **401 Unauthorized**   

Whereas a **successful**request will return a **200 OK**. 

Once completed, login to the account with the password.

**Question #2: Reset Jim's password!**

Believe it or not, the reset password mechanism can also be exploited! When inputted into the email field in the Forgot Password page, Jim's security question is set to *"Your eldest siblings middle name?"*.

In Task 2, we found that Jim might have something to do with **Star Trek**. Googling "Jim Star Trek" gives us a wiki page for **Jame T. Kirk** from Star Trek.

A screenshot of a computer

Description automatically generated

Looking through the wiki page we find that he has a brother.

A close up of a person's name

Description automatically generated

Looks like his brother's middle name is **Samuel**

Inputting that into the Forgot Password page allows you to successfully change his password.

You can change it to anything you want!

A screenshot of a login form

Description automatically generated

1. **AH! Don't look! -** [**Sensitive Data Exposure**](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure)

A screen shot of a computer screen

Description automatically generated

A web application should store and transmit sensitive data safely and securely. But in some cases, the developer may not correctly protect their sensitive data, making it vulnerable.

Most of the time, data protection is not applied consistently across the web application making certain pages accessible to the public. Other times information is leaked to the public without the knowledge of the developer, making the web application vulnerable to an attack.

More information: [Sensitive Data Exposure](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure)

**Question #1: Access the Confidential Document!**

A screenshot of a menu

Description automatically generated

Navigate to the **About Us** page, and hover over the *"Check out our terms of use"*.

A screen shot of a computer

Description automatically generated

You will see that it links to  [http://MACHINE\_IP/ftp/legal.md](http://machine_ip/ftp/legal.md). Navigating to that **/ftp/** directory reveals that it is exposed to the public!



A screenshot of a computer

Description automatically generated

We will download the **acquisitions.md**and save it. It looks like there are other files of interest here as well.

After downloading it, navigate to the **home page** to receive the flag!

**Question #2: Log into MC SafeSearch's account!**

After watching the video there are certain parts of the song that stand out.

He notes that his password is "**Mr. Noodles**" but he has replaced some "**vowels into zeros**", meaning that he just replaced the o's into 0's.

We now know the password to the *mc.safesearch@juice-sh.op* account is "**Mr. N00dles**"

Top of Form

**Question #3: Download the Backup file!**

We will now go back to the  [http://10.10.154.122/ftp/](http://machine_ip/ftp/) folder and try to download **package.json.bak**. But it seems we are met with a 403 which says that only .md and .pdf files can be downloaded.

A screenshot of a computer

Description automatically generated

To get around this, we will use a character bypass called "**Poison Null Byte**". A Poison Null Byte looks like this: ***%00***.

Note: as we can download it using the url, we will need to encode this 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!



**Why does this work?**

A Poison Null Byte is actually a **NULL terminator**. By placing a NULL character in the string at a certain byte, the string will tell the server to terminate at that point, nulling the rest of the string.

1. Who's flying this thing? -  [Broken Access Control](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control)

A screenshot of a computer

Description automatically generated

Modern-day systems will allow for multiple users to have access to different pages. Administrators most commonly use an administration page to edit, add and remove different elements of a website. You might use these when you are building a website with programs such as Weebly or Wix.

When Broken Access Control exploits or bugs are found, it will be categorised into one of **two types**:

|  |  |
| --- | --- |
| **Horizontal**Privilege Escalation | Occurs when a user can perform an action or access data of another user with the **same**level of permissions. |
| **Vertical**Privilege Escalation | Occurs when a user can perform an action or access data of another user with a **higher**level of permissions. |

A diagram of a broken access control

Description automatically generated

*Credits: Packetlabs.net*

More information: [Broken Access Control](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control)

**Question #1: Access the administration page!**

irst, we are going to open the **Debugger**on **Firefox**.

(Or **Sources** on **Chrome**.)

This can be done by navigating to it in the Web Developers menu.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

We are then going to refresh the page and look for a javascript file for **main-es2015.js**

We will then go to that page at: [http://MACHINE\_IP](http://machine_ip/)/main-es2015.js

A screenshot of a computer

Description automatically generated

To get this into a format we can read, click the { } button at the bottom  A search bar with a magnifying glass and a eye

Description automatically generated

Now search for the term "admin"

You will come across a couple of different words containing "admin" but the one we are looking for is "path: administration"

A screenshot of a computer

Description automatically generated

This hints towards a page called "**/#/administration**" as can be seen by the **about**path a couple lines below, but going there while not logged in doesn't work.

As this is an Administrator page, it makes sense that we need to be in the **Admin account**in order to view it.

A good way to stop users from accessing this is to only load parts of the application that need to be used by them. This stops sensitive information such as an admin page from been leaked or viewed.

**Question #2: View another user's shopping basket!**

Login to the Admin account and click on 'Your Basket'. Make sure Burp is running so you can capture the request!

Forward each request until you see: *GET /rest/basket/1 HTTP/1.1*

A screenshot of a computer

Description automatically generated

Now, we are going to change the number **1** after /basket/ to **2**



It will now show you the basket of UserID 2. You can do this for other UserIDs as well, provided that they have one!

A screenshot of a menu

Description automatically generated

Top of Form

**Question #3: Remove all 5-star reviews!**

Navigate to the [http://MACHINE\_IP/#/administration](http://machine_ip/#/administration) page again and click the bin icon next to the review with 5 stars!

A screenshot of a customer feedback

Description automatically generated

1. Where did that come from? - [Cross-Site Scripting XSS](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS))

**There are three major types of XSS attacks:**

|  |  |
| --- | --- |
| DOM (Special) | **DOM XSS** *(Document Object Model-based Cross-site Scripting)* uses the HTML environment to execute malicious javascript. This type of attack commonly uses the *<script></script>* HTML tag. |
| Persistent (Server-side) | **Persistent XSS** is javascript that is run when the server loads the page containing it. These can occur when the server does not sanitise the user data when it is **uploaded**to a page. These are commonly found on blog posts. |
| Reflected (Client-side) | **Reflected XSS** is javascript that is run on the client-side end of the web application. These are most commonly found when the server doesn't sanitise **search**data. |

**Question #1: Perform a DOM XSS!**



We will be using the iframe element with a javascript alert tag:

*<iframe src="javascript:alert(`xss`)">*

Inputting this into the **search bar** will trigger the alert.

A screenshot of a computer

Description automatically generated

Note that we are using **iframe**which is a common HTML element found in many web applications, there are others which also produce the same result.

This type of XSS is also called XFS (Cross-Frame Scripting), is one of the most common forms of detecting XSS within web applications.

Websites that allow the user to modify the iframe or other DOM elements will most likely be vulnerable to XSS.

**Why does this work?**

It is common practice that the search bar will send a request to the server in which it will then send back the related information, but this is where the flaw lies. Without correct input sanitation, we are able to perform an XSS attack against the search bar.

**Question #2: Perform a persistent XSS!**

First, login to the **admin**account.

We are going to navigate to the "**Last Login IP**" page for this attack.

A screenshot of a computer

Description automatically generated

It should say the last IP Address is 0.0.0.0 or 10.x.x.x

As it logs the 'last' login IP we will now logout so that it logs the 'new' IP.



Make sure that Burp **intercept is on**, so it will catch the logout request.

We will then head over to the Headers tab where we will add a new header:

|  |  |
| --- | --- |
| *True-Client-IP* | *<iframe src="javascript:alert(`xss`)">* |

A screenshot of a computer

Description automatically generated

Then forward the request to the server!  
When **signing back into the admin account** and navigating to the Last Login IP page again, we will see the XSS alert!

A screenshot of a computer

Description automatically generated

**Why do we have to send this Header?**

The *True-Client-IP*header is similar to the *X-Forwarded-For*header, both tell the server or proxy what the IP of the client is. Due to there being no sanitation in the header we are able to perform an XSS attack.

**Question #3: Perform a reflected XSS!**

First, we are going to need to be on the right page to perform the reflected XSS!

**Login**into the **admin account** and navigate to the '**Order History**' page.

A screenshot of a computer

Description automatically generated

From there you will see a "**Truck**" icon, clicking on that will bring you to the track result page. You will also see that there is an id paired with the order.   

We will use the iframe XSS, *<iframe src="javascript:alert(`xss`)">,*in the place of the *5267-f73dcd000abcc353*

After submitting the URL, refresh the page and you will then get an alert saying XSS!

A screenshot of a computer

Description automatically generated

**Why does this work?**

The server will have a lookup table or database (depending on the type of server) for each tracking ID. As the 'id' parameter is not sanitised before it is sent to the server, we are able to perform an XSS attack.