TEST REPORT

TEST TARGET

Car rental application hosted on the AWS cloud.

SECURITY TEAM

Sumanth Vankineni
Dhanush Arvind

LIST OF FIGURES

page

TABLE OF CONTENTS

Page

1. INTRODUCTION

The car rental application can be accessed with the following IP address
http://44.66.88.99/carrental

The following penetration testing report is a comprehensive and detailed analysis of the security posture of an application that our team built and deployed on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance along with Amazon RDS which is the managed database that allows us to host, create and modify relational databases like Mongo DB, MySQL, and others. In our case we used the MySQL database as the relational database to host the application. The objective of this pen test was to identify any security weaknesses, vulnerabilities, and threats that could compromise the CIA triad (confidentiality, integrity, and availability of the application).

The testing methodology included a combination of manual and automated techniques and tools to simulate real-world attack scenarios. We conducted extensive testing and analysis of various aspects of the application, including its infrastructure, application code, authentication and authorization mechanisms, and data storage and transmission methods. This application is a simple car rental portal for clients to rent exclusive cars from the company. The website was created purely for users to book their rental car through the website and more about the website will be mentioned during the build phase.

This report provides a detailed overview of the vulnerabilities discovered during the penetration testing and recommendations for remediation to improve the security aspect of the application. The application was built using php. A lot of security measures weren't considered assuming the company had limited budget during the software development life cycle of the application.

2. Build Phase

As mentioned before, the application was built using php and MySQL database to store user credentials, available vehicles, and its details.

2.1 STEP 1:

During the building phase of the PHP application, our team tried to follow best practices to ensure the application's security. The main aspect of the application was security, our team used secure coding practices to minimize the risk of vulnerabilities such as SQL injection and CSRF. Additionally, the team implemented authentication and access control mechanisms to ensure that only authorized users can access sensitive data and functionality. The authentication part of the application was implemented once it was hosted to the AWS cloud.

2.2 STEP 2:

The next part of the application was to host the application outside of AWS and test if all the components of the application were working. (Note: The application is a simple website and not industry level, with a limited set of functionalities and features. We hosted the application using XAAMP automated software to launch the Apache software and then used MySQL server to connect the SQL file of the application.

2.3 STEP 3:

The next step was to migrate the entire application to the cloud and then host the application using the AWS EC2 instance along with AWS RDS. While hosting on the cloud platform a lot of configurations had to be changed. The reason to host the application on the cloud was strictly to implement more security measures to the application leveraging the in-built tools that comes with AWS platform. Anyone can access the application from the link given in page 3.

3. VULNERABILITY ASSESSMENT

3.1 STEP 1: Nmap Scan

3.1.1 VULNERABILITY 1

- The first step of the assessment was to enumerate the host using the most common scanning toll Nmap.

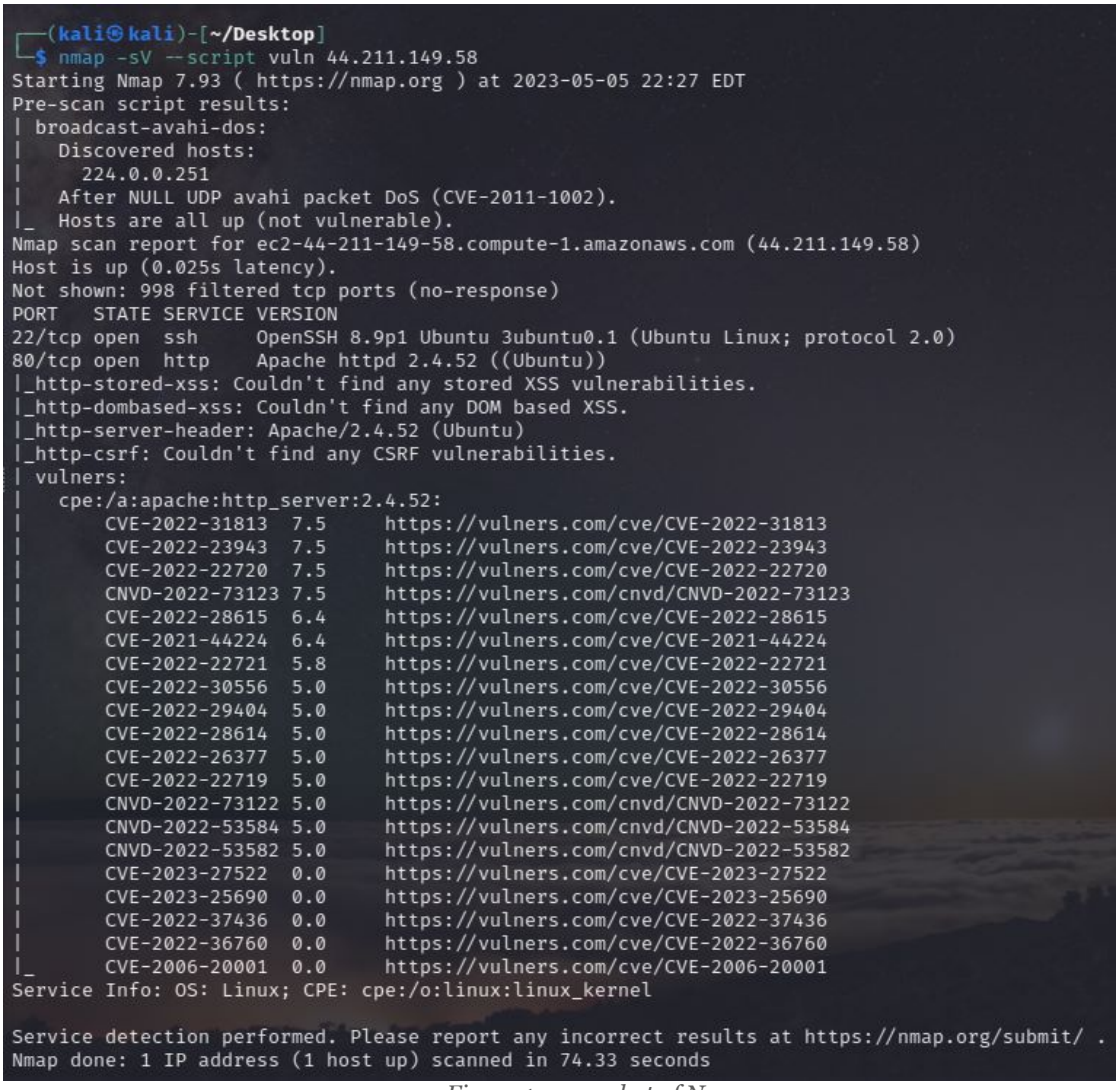


Figure 1 screenshot of Nmap scan

- When we ran the Nmap scan on the host, we discovered possible vulnerabilities that the application might be susceptible to. The vulnerabilities and their CVEs are listed in the Nmap scan in the above screenshot. The open ports are 22 and 80, indicating that secure shell is open for users to connect using their private key or password. The SSH version is the latest, and no vulnerabilities have been discovered in the current version.

- Port 80 is also open, indicating that a website is hosted on the server. One of the vulnerabilities associated with this version of httpd is "HTTP Request Smuggling

against backend web applications," as indicated by CVE-2022-22720. This vulnerability allows attackers to bypass security controls implemented in web application firewalls (WAFs) and other security devices to attack backend web applications. This vulnerability has a base score of 9.8 and is considered critical.

- One way to mitigate this vulnerability is to use the updated version of httpd that comes with the EC2 instance.

3.2 STEP 2: Directory discovery using GoBuster.

3.2.1 VULNERABILITY 2

- The next step of enumeration is to find out all the directory listings in the web application. To find all the directories in the web application, we used gobuster and ran it against the server.

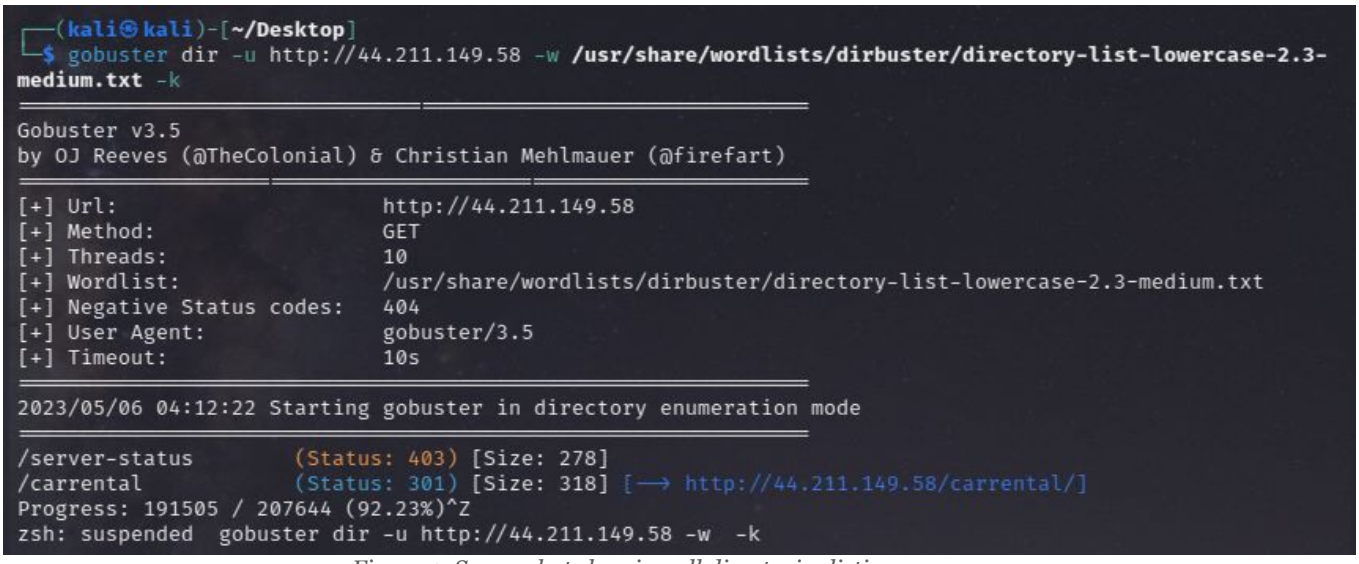


Figure 2 Screenshot showing all directories listing.

- The screenshot above shows all the directories that were listed when we ran the Gobuster tool against the public IP address of the application hosted on the EC2.

- We see that the "carrental" directory takes us to the web application and displays the index.php homepage to the users.

- We then ran a Gobuster scan within the carrental directory.

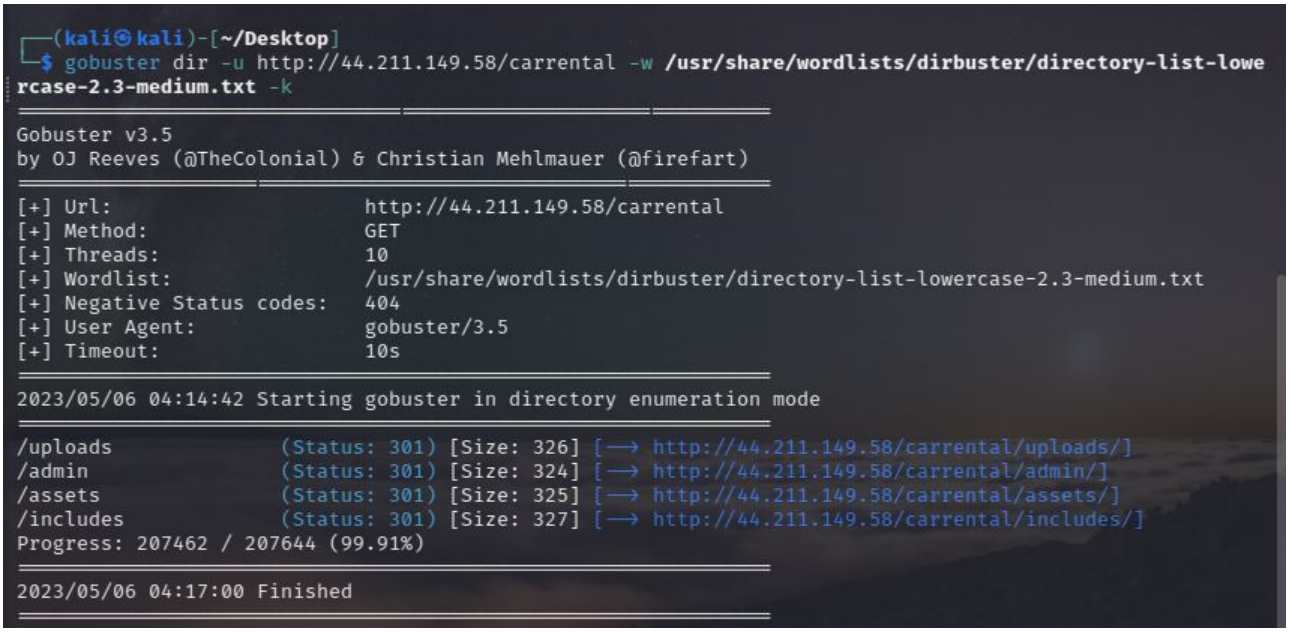


Figure 3 Screenshot of all directories using Gobuster

- However, we then discovered directories that are not intended to be accessed by the public, which presents a potential vulnerability. All the directories listed in the "carrental" directory are being displayed and can be accessed by the public (status: 301 -> redirected). To mitigate this, it is important to restrict access to these directories and ensure that they are not displayed to unauthorized users.

3.2.2 MITAGATION

- **Access control**: Ensure that people who can access secret.txt file is restricted to authorized users only. Use authentication mechanisms such as usernames and passwords, or IP-based access controls to restrict access to the file.
- **Using HTTPS instead of HTTP**: with HTTPS the data while in transmission won't be intercepted by attackers and cannot have access to confidential data which was not intended for anyone.
- **File permission and file access**: This is the most important mitigation strategy when it comes to file access through the website. Giving appropriate file permissions and limiting access to all the users who are not supposed to be accessing certain directories and files is very important.

  One of the ways to restrict users accessing these files is by creating the ".htaccess" file to set appropriate permissions for users to limit access.

```
ubuntu@ip-172-31-92-152:/var/www/html/carrental/assets$ ls -la
total 32
drwxrwxr-x 7 ubuntu ubuntu 4096 May  7 00:01 .
drwxrwxr-x 8 ubuntu ubuntu 4096 May  6 17:58 ..
-rw-rw-r-- 1 ubuntu ubuntu  221 May  7 00:01 .htaccess
drwxrwxr-x 2 ubuntu ubuntu 4096 May  2 03:34 css
drwxrwxr-x 2 ubuntu ubuntu 4096 May  2 03:34 fonts
drwxrwxr-x 3 ubuntu ubuntu 4096 May  2 03:34 images
drwxrwxr-x 2 ubuntu ubuntu 4096 May  2 03:34 js
drwxrwxr-x 4 ubuntu ubuntu 4096 May  2 03:34 switcher
ubuntu@ip-172-31-92-152:/var/www/html/carrental/assets$
```

Figure 4 Screenshot showing the .htaccess file creation.

The following screenshot shows creation of .htaccess file and then giving appropriate permissions to users.

```
Order deny,allow
Deny from all
Allow from 192.168.127.129

# Display a custom message to unauthorized users
ErrorDocument 403 "Sorry, access to this directory is restricted."

# Prevent directory listing
Options -Indexes
~
~
```

Figure 5 Screenshot of the .htaccess file to deny permission.

In the .htaccess file, any user except the given IP address (192.168.127.129) cannot access this file through the browser. To cross check if the strategy worked, we would run the gobuster scan one more time to check.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ gobuster dir -u http://44.211.149.58/carrental -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2
.3-medium.txt -k
===============================================================
Gobuster v3.5
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://44.211.149.58/carrental
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.5
[+] Timeout:                 10s
===============================================================
2023/05/06 20:01:29 Starting gobuster in directory enumeration mode
===============================================================
/uploads              (Status: 301) [Size: 326] [--> http://44.211.149.58/carrental/uploads/]
/admin                (Status: 301) [Size: 324] [--> http://44.211.149.58/carrental/admin/]
/assets               (Status: 403) [Size: 46]
/includes             (Status: 301) [Size: 327] [--> http://44.211.149.58/carrental/includes/]
Progress: 24509 / 207644 (11.80%)^Z
zsh: suspended  gobuster dir -u http://44.211.149.58/carrental -w  -k
```

Figure 6 Screenshot of the .htaccess implementation

As shown in the above screen shot, there is an intended error message we desire to display when anyone tried to access the “assets” directory through the web. This was just for demonstration purposes, as most of the images are being called from this directory.

3.3 STEP 3: PHP File Upload

3.3.1 VULNERABILITY_3

- The next step would be to browse the website to check for any vulnerabilities in the web application.
- As we browsed through the website, we came across a path where we could upload a .txt file to the webserver to get the clients documents before renting a high-end car. This was necessary for some of the important exclusive cars that the clients wanted to rent. We just implemented a simple php code which is being called to upload a document “.txt” to the web server.

- So let’s try uploading a simple .txt file that we could create and see if we could access the file.

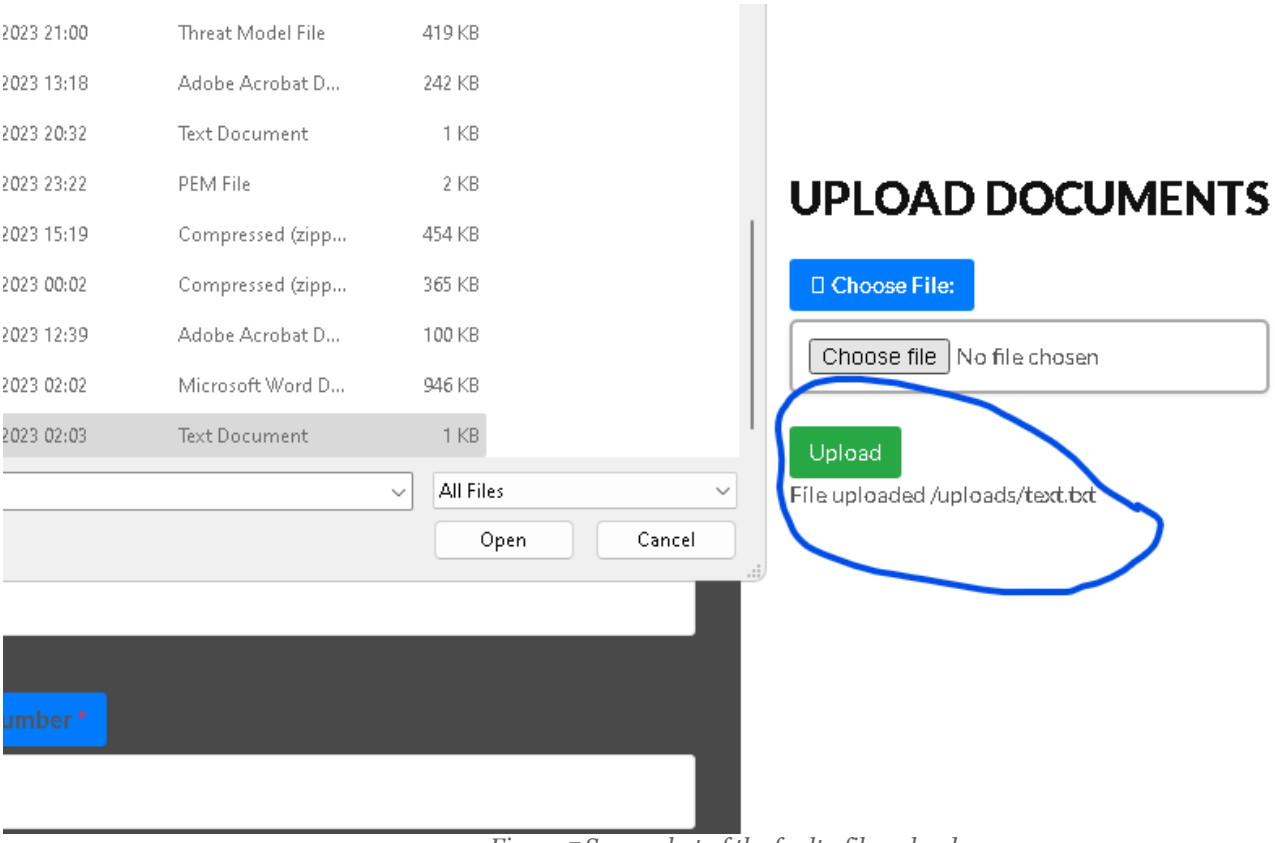


Figure 7 Screenshot of the faulty file upload

- Seems like there was no problem uploading a file to the webserver, the next step would be to check if the following .txt file can be accessed through the browser. If this is possible, this could be a possible vulnerability.

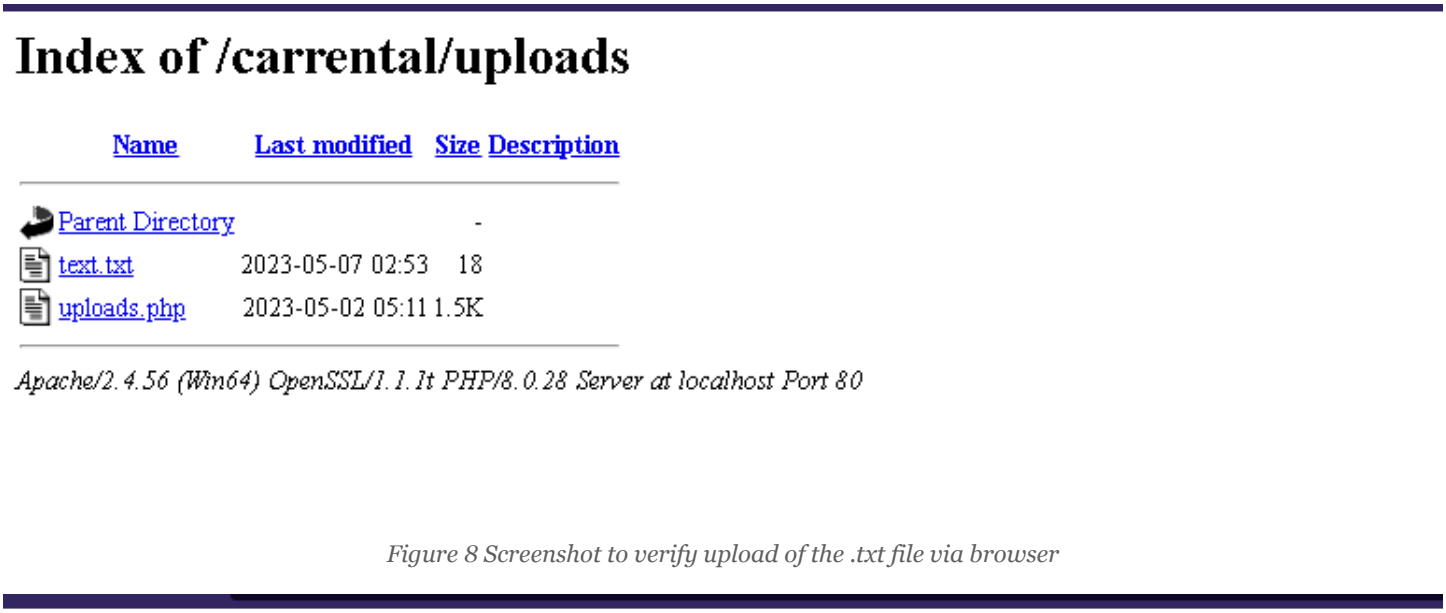


Figure 8 Screenshot to verify upload of the .txt file via browser

Figure 9 screenshot of the .txt file

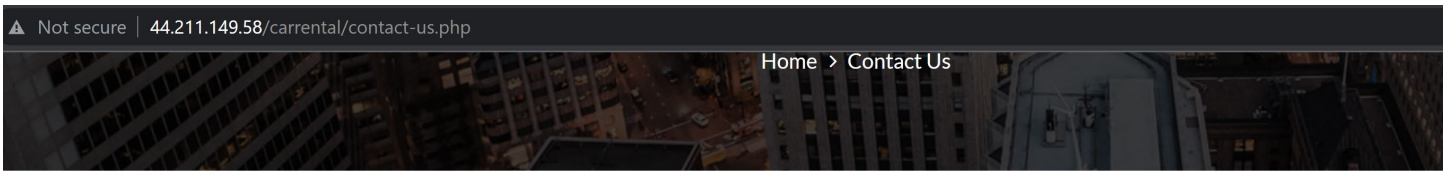
- This directory being accessed by the public is very critical and dangerous. This again could use ".htaccess" file to limit the access of the directory, hence the code is also stored in the same directory, it is recommended to store all the uploaded files in a separate directory and restrict access to public use.

- Now the next step would be to upload any php file and see if that file gets uploaded. Since only txt files are supposed to be uploaded, it is recommended to sanitize the inputs.

- Before uploading the php file we will put malicious code into the file and check if that command executes once uploaded by the user.

Figure 10 Screenshot of the malicious script

This code if it can be uploaded, will run the command “whoami”, and then display the current user of the system.



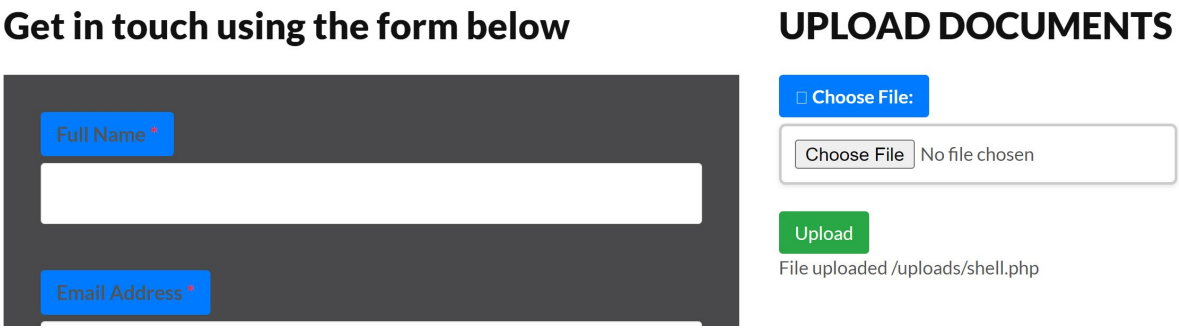


Figure 11 Screenshot of the php file upload

Looks like the malicious php did indeed get uploaded, and this is a very dangerous vulnerability when it comes to websites that allow any file to be uploaded to the server.

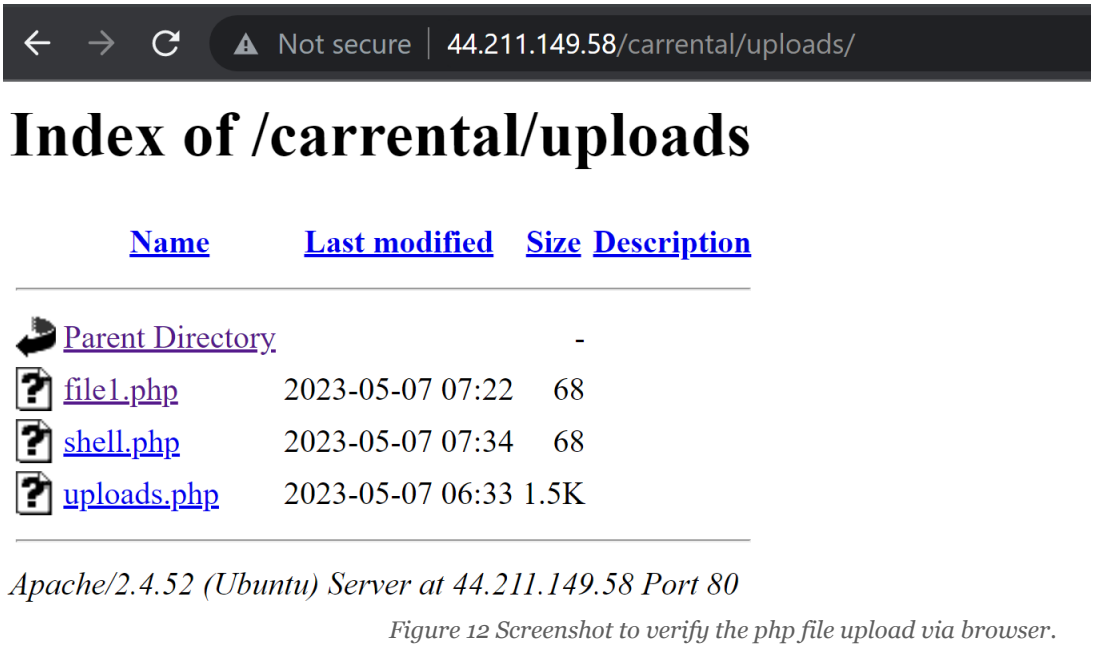


Figure 12 Screenshot to verify the php file upload via browser.

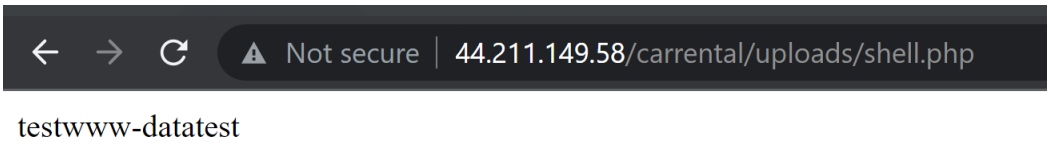


Figure 13 Screenshot to verify the code execution.

This shows that the file once executed will run the command and then display the intended results. As the screen shot displays, we are running as www-data on the system.

3.3.2 MITIGATION

There are many ways to mitigate this issue, we can also implement multiple strategies to mitigate this vulnerability.

3.3.2.1 STRATEGY 1

- One method would be to limit access to the folder where the uploaded files are stored.
- In our situation, the "upload.php" file is being run from the same location where all the files are stored, which is not considered a good coding practice.
- It is recommended to create a separate folder for all uploaded files on the EC2 instance to restrict public access to this folder and prevent others from running the code on their own.
- However, this approach comes with a problem. Even if the attacker cannot run the script manually on their system, they could still upload a "reverse_tcp" shell

and wait for anyone from the company to open the script. Then, they could run the command and the attacker could gain access to the system.

3.3.2.2 STRATEGY 2

- This method would solve the previous problem. Sanitizing the file format and restricting any php, or other extensions which would bring harm to the system would be the best idea when it comes to mitigating file upload.

```
<?php

// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
      $target_dir = "uploads/";
      $target_file = $target_dir . basename($_FILES["file"]["name"]);

    move_uploaded_file($_FILES["file"]["tmp_name"], $target_file);
    echo "File uploaded /uploads/".$_FILES["file"]["name"];
}
?>
```

Figure 14 Screenshot of code to restrict php file upload.

- This is the current PHP code that is being called when files are uploaded from the browser to the web server. A secure way of coding would be to limit the allowed file extensions to those that can be safely uploaded to the server, rather than allowing intrusive extensions like '.php'."

```
<?php

if(isset($_POST["submit"])) {
      $target_dir = "uploads/";
      $target_file = $target_dir . basename($_FILES["file"]["name"]);
      $uploadOk = 1;
      $imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
      $type = $_FILES["file"]["type"];

   if($_FILES["file"]["type"] != "text/plain") {
    echo "Only text files (.txt) are allowed.";
    $uploadOk = 0;
}

    if($uploadOk == 1){
        move_uploaded_file($_FILES["file"]["tmp_name"], $target_file);
        echo "File uploaded /uploads/".$_FILES["file"]["name"];
    }
}
?>

</body>
</html>
```

Figure 15 screenshot showing code to restrict any php file upload.

- This should solve the upload problem almost completely; it is recommended to sanitize the extension and allow only certain extensions to be uploaded to the webserver. Here, the allowed extension is only ".txt" file format and nothing else.

Finally, there are other strategies that could be implemented. One such strategy is hosting the web application on a Windows server. This could make it harder for attackers to exploit the vulnerability, as Windows machines provide certain security measures against bad coding practices by default and do not allow malicious code to run on the system. However, it should be noted that this solution may not completely solve the problem.

3.4 STEP 4: CROSS SITE SCRIPTING

3.4.1 VULNERABILITY 4

- As we browse through the website, we come across a search bar. The search bar searches for the car that is in the inventory and displays if the car is available to rent or not.

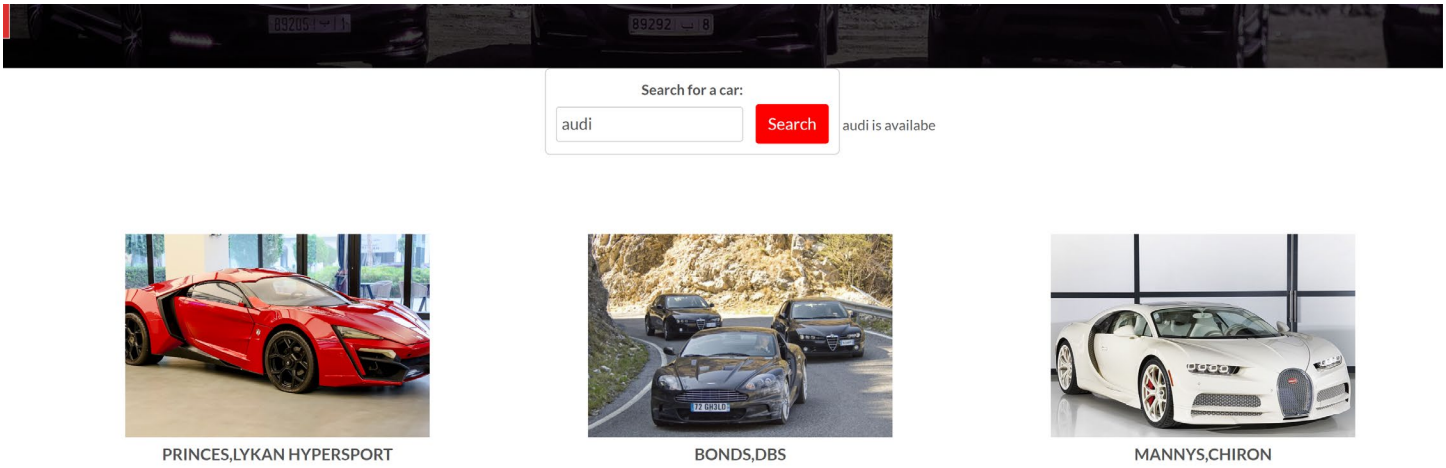


Figure 16 Screenshot of the site search bar

- This searches for the car and displays if the car is available for rent or not as shown in the screenshot.

- Let's check if this search bar is vulnerable to XSS or not.

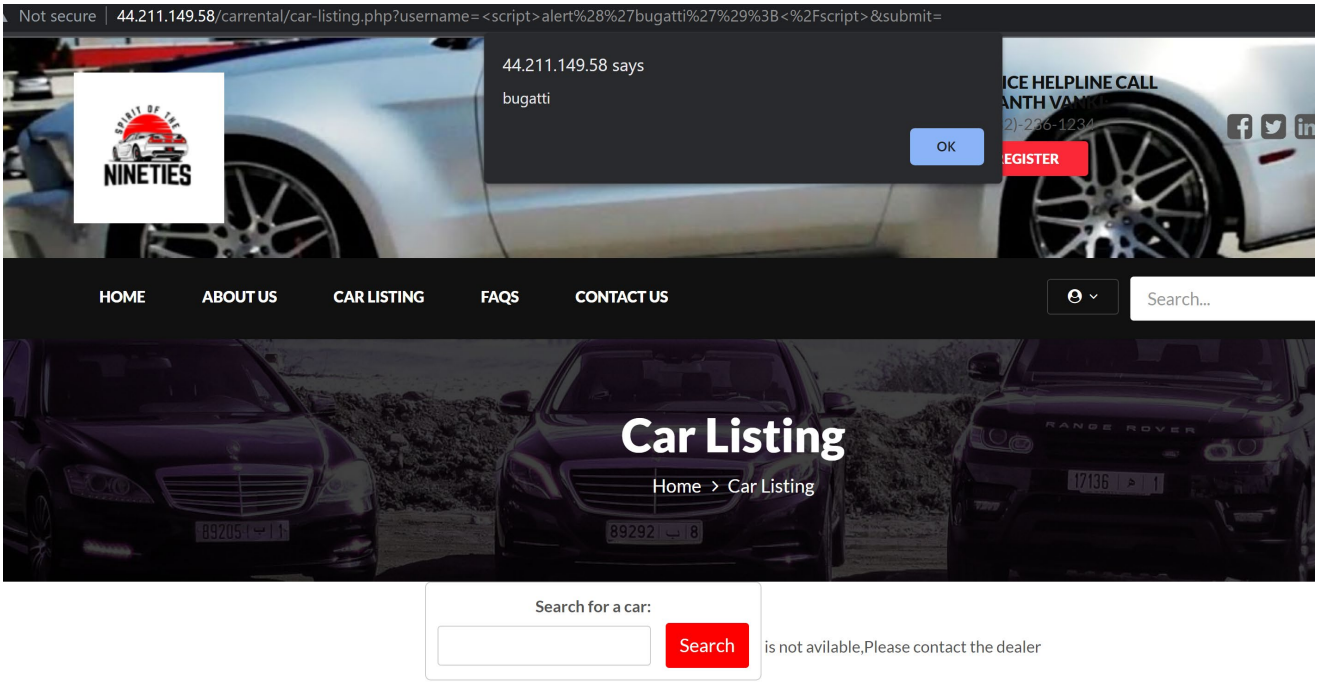


Figure 17 Screenshot of the website vulnerable to XSS

- As the screenshot shows, the server responds with whatever input we provide. In this case, I entered the code '<script>alert("Bugatti")</script>' into the search bar, which was a simple script message to the server that executed the command. This confirms that the website is vulnerable to cross-site scripting attacks.

3.4.2 MITIGATION

- **Input validation**: Validate and sanitize all user input, in our case sanitizing the user from inputting any special characters into the search bar must be implemented. This will reduce the chance of XSS attacks on the web server.

- **Output encoding**: Encode all output to ensure that any user input is displayed as plain text rather than being interpreted as HTML or JavaScript code. This can be done using encoding functions like html special chars or html entities.

- **Content Security Policy (CSP)**: Implement a CSP header in the HTTP response that restricts the types of content that the website can load, such as scripts and stylesheets. This can help prevent malicious code injection.

```php
<?php
$cars= array("audi","bugatti","bmw","polo gt");
if(isset($_GET["search"])){
  $username = $_GET["search"];
  $found=false;
  foreach($cars as $a){
      if($a == $username){
         $found = true;
         break;
    }
  }
  if($found) {
    echo $username . " is availabe";
    $color="green";
  } else {
    echo $username . " is not avilable,Please contact the dealer";
    $color="green";
  }
}

?>
```

Figure 18 Screenshot of the code without modification

- This is the current code that runs when searching for cars. To increase security, we could sanitize the input by disallowing certain characters from being executed.

```php
<?php
$cars= array("audi","bugatti","bmw","polo gt");
if(isset($_GET["search"])){
  $username = preg_replace("/<(.*)[S,s](.*)[C,c](.*)[R,r](.*)[I,i](.*)[P,p](.*)[T,t]>/i", "", $_GET["search"]);
  $found=false;
  foreach($cars as $a){
      if($a == $username){
         $found = true;
         break;
    }
  }
  if($found) {
    echo $username . " is availabe";
    $color="green";
  } else {
    echo $username . " is not avilable,Please contact the dealer";
    $color="green";
  }
}

?>
<style>
```

Figure 19 Screenshot of the modified code

- This modified code will replace certain characters with an empty string, preventing the execution of some characters. This will prevent the entire code from being executed and mitigate XSS attacks.

- This does not guarantee complete prevention of XSS attacks since there are ways to bypass restrictions by using certain characters such as polyglots. The use of polyglots will bypass certain kinds of restrictions that come while trying to exploit the web application with Cross Site Scripting.

STEP 5: WEAK DEFAULT CREDENTIALS

3.5 VULNERABILITY 5

Another directory we discovered is the admin portal. This page only takes two inputs from the user. The username and the password.

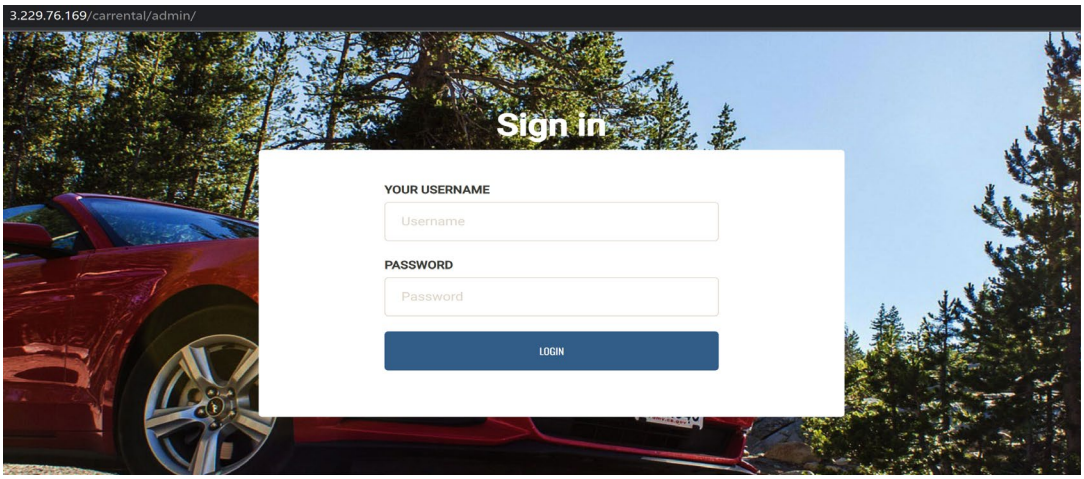


Figure 20 Screenshot of the Admin Portal

- Keeping weak and default credentials in the admin page is a major concern when it comes to system breaches and data exposure. Many startups tend to leave the default credentials unchanged, which is not a good practice, especially during software development and release. We wanted to perform a brute force attack on the admin page using multiple wordlists to check for any vulnerabilities.

- We utilized Burp Suite to execute the brute force attack on the admin page.

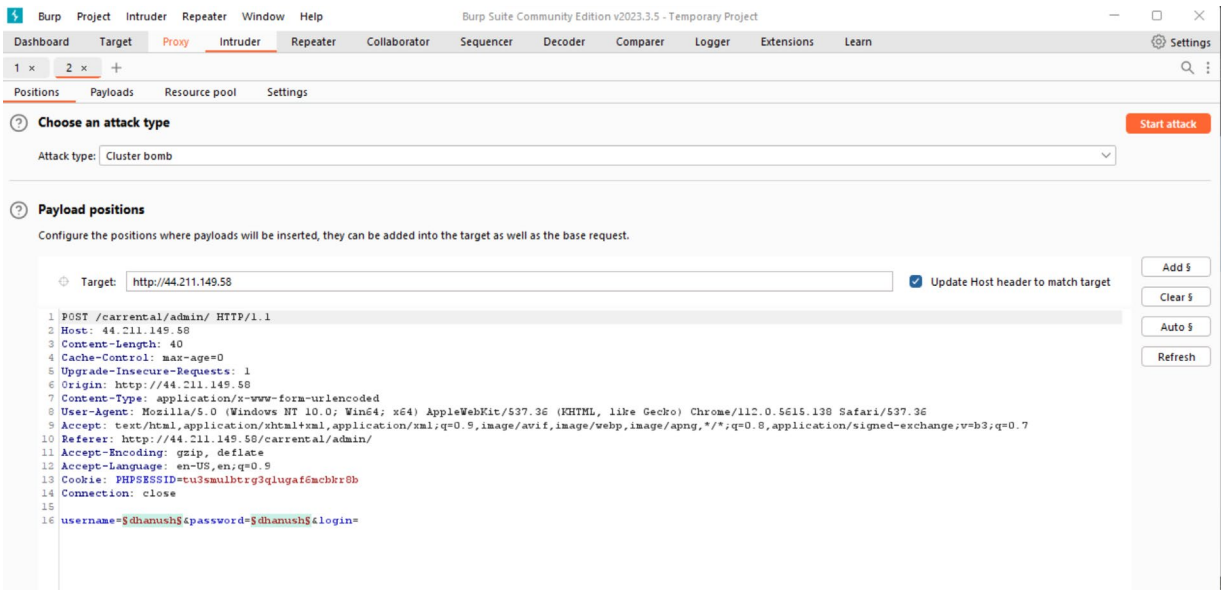


Figure 21 Screenshot of the packet captured using Burp suite.

- We captured the data packet using the wrong credentials and then sent it to intruder to start the bruteforce attack.

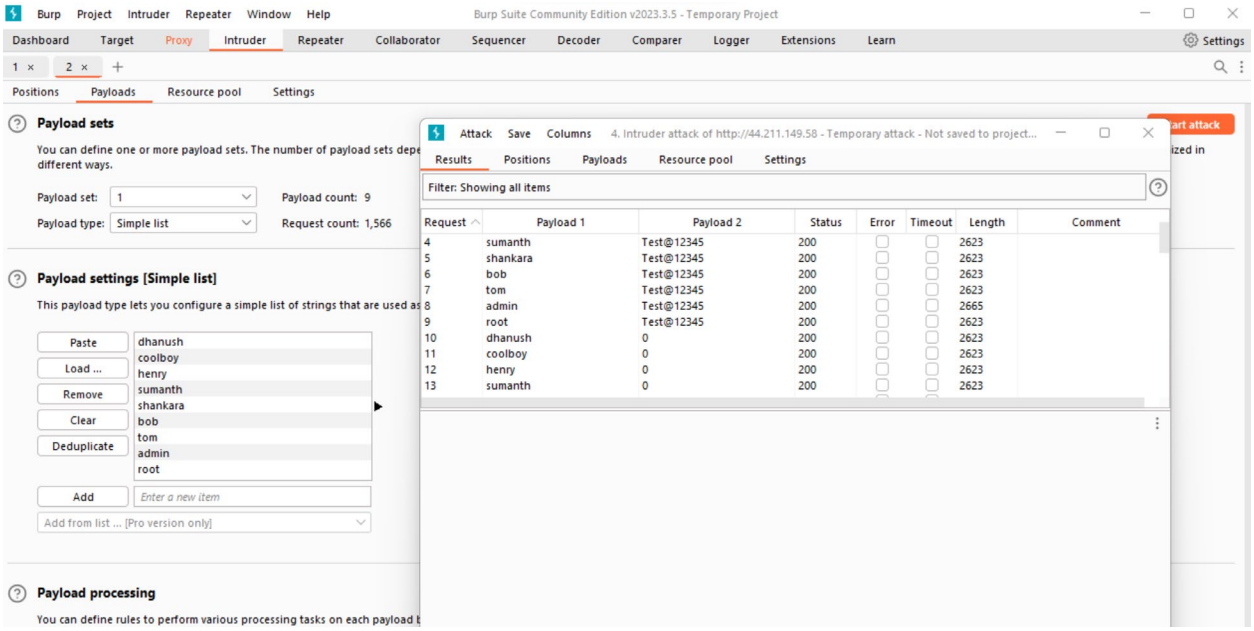
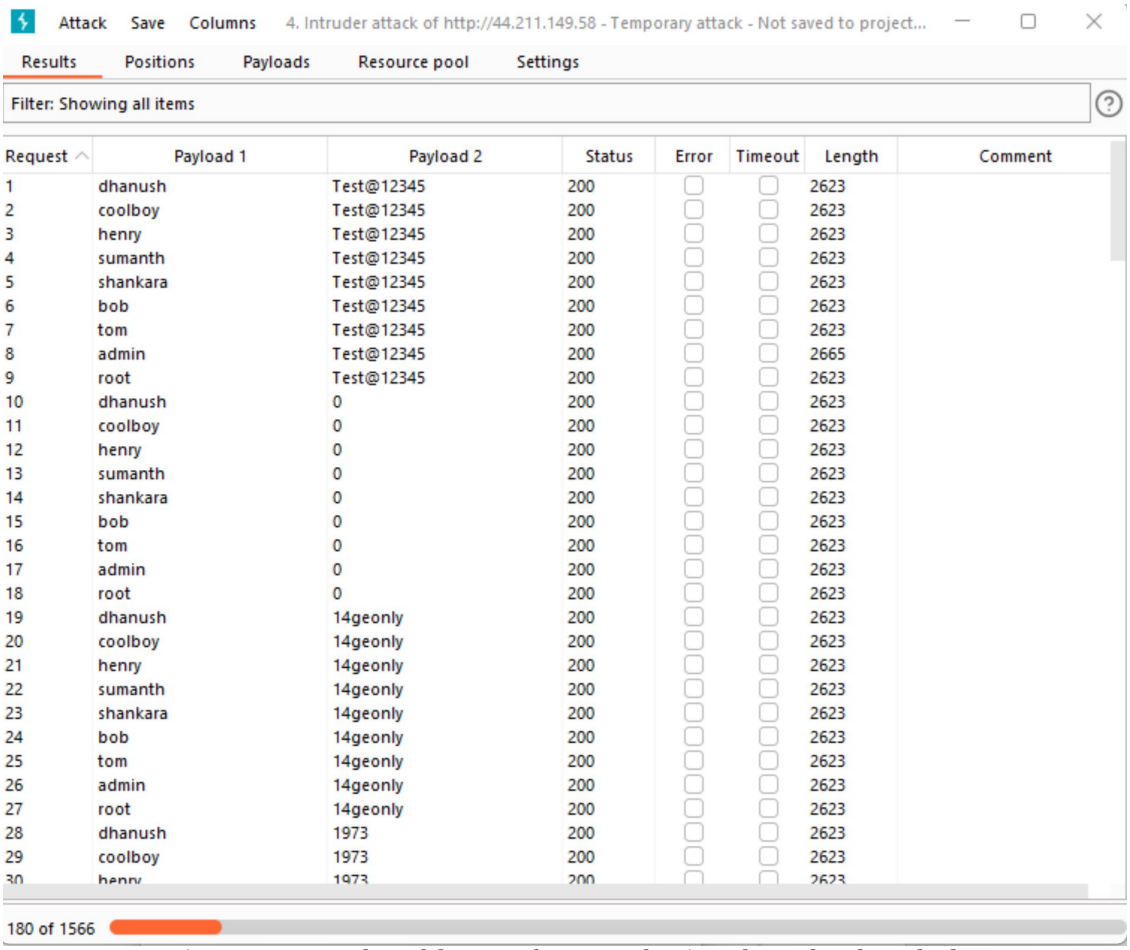


Figure 22 Screenshot of the brute force attack

- The attack method we chose was the cluster bomb attack to bruteforce each word in one wordlist with all the words in the other.

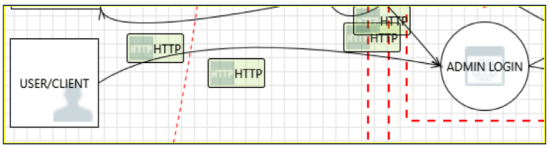


Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
1	dhanush	Test@12345	200			2623	
2	coolboy	Test@12345	200			2623	
3	henry	Test@12345	200			2623	
4	sumanth	Test@12345	200			2623	
5	shankara	Test@12345	200			2623	
6	bob	Test@12345	200			2623	
7	tom	Test@12345	200			2623	
8	admin	Test@12345	200			2665	
9	root	Test@12345	200			2623	
10	dhanush	0	200			2623	
11	coolboy	0	200			2623	
12	henry	0	200			2623	
13	sumanth	0	200			2623	
14	shankara	0	200			2623	
15	bob	0	200			2623	
16	tom	0	200			2623	
17	admin	0	200			2623	
18	root	0	200			2623	
19	dhanush	14geonly	200			2623	
20	coolboy	14geonly	200			2623	
21	henry	14geonly	200			2623	
22	sumanth	14geonly	200			2623	
23	shankara	14geonly	200			2623	
24	bob	14geonly	200			2623	
25	tom	14geonly	200			2623	
26	admin	14geonly	200			2623	
27	root	14geonly	200			2623	
28	dhanush	1973	200			2623	
29	coolboy	1973	200			2623	
30	henry	1973	200			2623	

Figure 23 Screenshot of the Bruteforce attack using Cluster bomb method.

- As seen in the above figure, the “admin” with password “Test@12345” gave a different length value compared to all the other pairs. When tested, it worked, and the user could login using these credentials.

Interaction: HTTP



39. Spoofing the ADMIN LOGIN Process [State: Not Started] [Priority: High]

Category: Spoofing
Description: ADMIN LOGIN may be spoofed by an attacker and this may lead to information disclosure by USER/CLIENT. Consider using a standard authentication mechanism to identify the destination process.
Justification: <no mitigation provided>

40. Potential Lack of Input Validation for ADMIN LOGIN [State: Not Started] [Priority: High]

Category: Tampering
Description: Data flowing across HTTP may be tampered with by an attacker. This may lead to a denial of service attack against ADMIN LOGIN or an elevation of privilege attack against ADMIN LOGIN or an information disclosure by ADMIN LOGIN. Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.
Justification: <no mitigation provided>

Figure 24 Screenshot of the threat modelling

This screenshot when the threat model report was created talks about the potential lack of input validation for “ADMIN LOGIN” page. This is one of the most important vulnerabilities to take a careful look at and mitigate this vulnerability.

3.5.1 MITIGATION

To mitigate the risk of weak admin page credentials being susceptible to brute force attacks, we can implement the following measures:

- **Strong Password Policies**: Enforcing strict passwords, these include a combination of uppercase and lowercase letters, numbers, and special characters. Discourage the use of common or easily guessable passwords.

- **Password Complexity Checks**: Implement a mechanism to validate the strength of the passwords during user registration or password change processes. This can include checks for minimum length, character diversity, and avoidance of common patterns.

3.6 AWS CLOUD VULNERABILITIES

- During the penetration test, we employed Prowler, a security tool designed for AWS environments, to identify misconfigurations and security vulnerabilities. Prowler allowed us to thoroughly assess the AWS infrastructure and highlight potential areas of concern. By conducting a comprehensive analysis of the configuration settings, permissions, and access controls within the AWS environment, we were able to identify any potential weaknesses that could have been exploited by malicious actors.

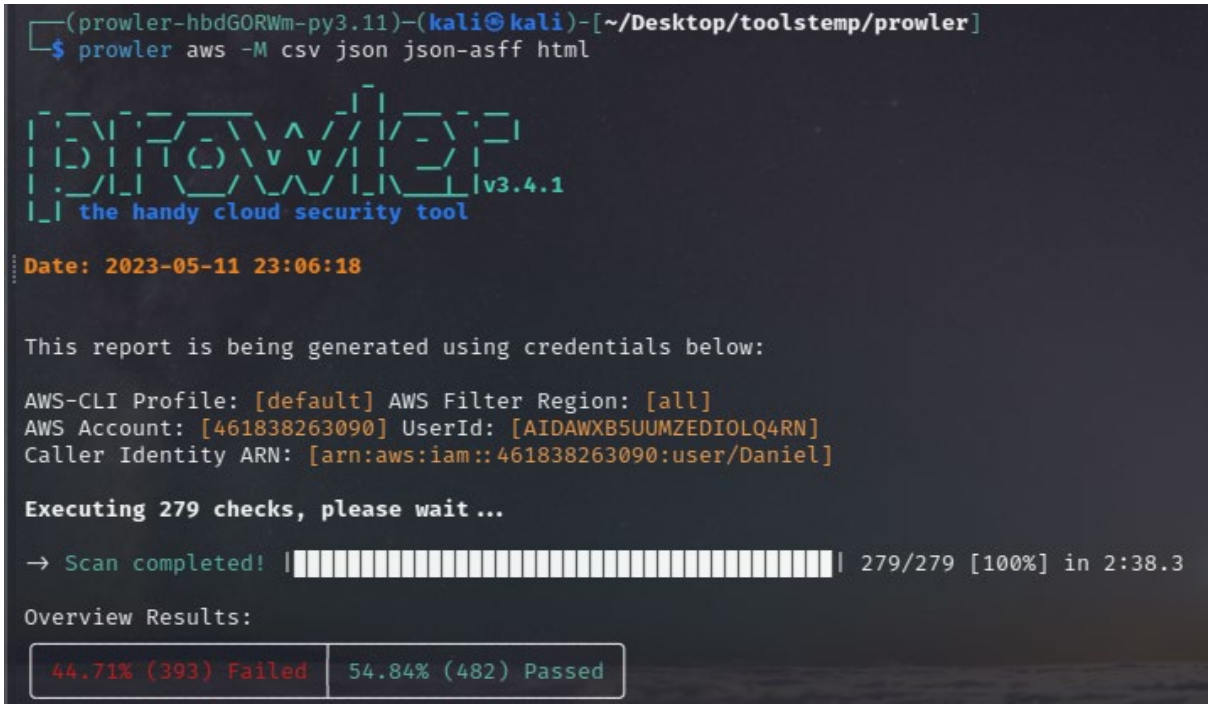


Figure 25 Screenshot of AWS Vulnerabilities using Prowler.

Account 461838263090 Scan Results (severity columns are for fails only):

Provider	Service	Status	Critical	High	Medium	Low
aws	accessanalyzer	FAIL (34)	0	0	0	34
aws	account	PASS (3)	0	0	0	0
aws	backup	FAIL (2)	0	0	0	2
aws	cloudtrail	FAIL (19)	0	17	0	2
aws	cloudwatch	FAIL (15)	0	0	15	0
aws	config	FAIL (17)	0	0	17	0
aws	drs	FAIL (17)	0	0	17	0
aws	ec2	FAIL (113)	0	32	81	0
aws	emr	PASS (17)	0	0	0	0
aws	glue	FAIL (34)	0	0	34	0
aws	iam	FAIL (21)	0	5	10	6
aws	inspector2	FAIL (16)	0	0	16	0
aws	macie	FAIL (17)	0	0	0	17
aws	network-firewall	FAIL (17)	0	0	17	0
aws	organizations	FAIL (3)	0	0	2	1
aws	rds	FAIL (6)	0	0	5	1
aws	resourceexplorer2	FAIL (1)	0	0	0	1
aws	s3	FAIL (8)	0	1	6	1
aws	securityhub	FAIL (17)	0	0	17	0
aws	ssm	FAIL (1)	0	0	1	0
aws	trustedadvisor	PASS (1)	0	0	0	0
aws	vpc	FAIL (35)	0	0	35	0

* You only see here those services that contains resources.

Figure 26 Screenshot of AWS Vulnerabilities

The scan conducted using Prowler in the AWS environment revealed several potential vulnerabilities. These findings uncovered misconfigurations and security gaps that could have exposed sensitive data or provided unauthorized access to resources.

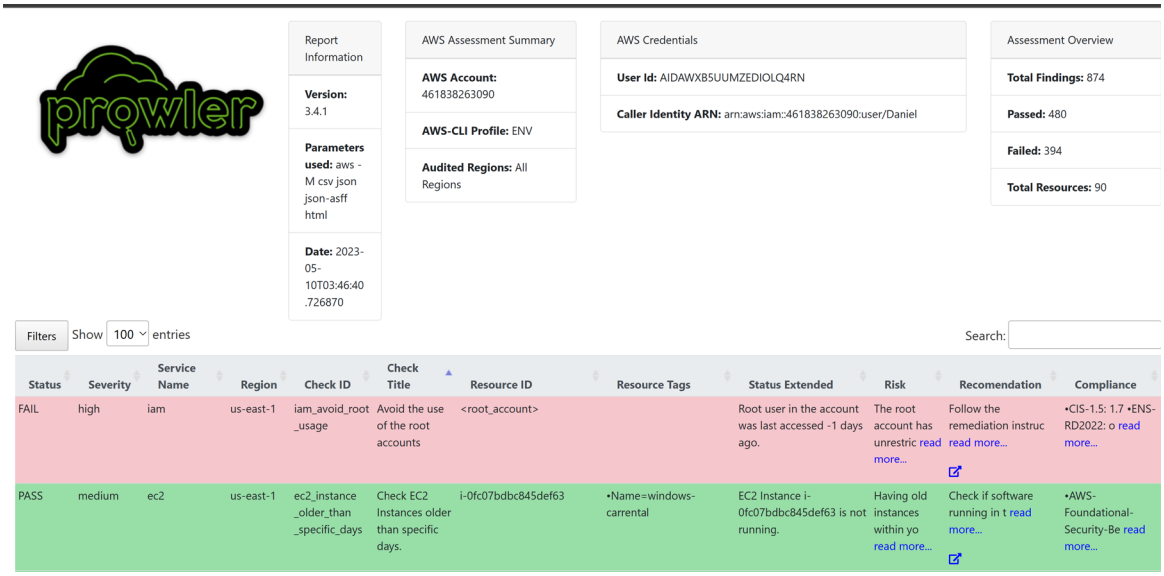


Figure 27 Screenshot of AWS Vulnerabilities

- One of the major concerns was the S3 bucket vulnerability. Where the security credentials are compromised or unauthorized access is granted.

- Another mitigation would be to Adding a Multi Factor Authentication (MFA) delete to an S3 bucket, requires additional authentication when you change the version state of your bucket, or you delete and object version adding another layer of security in the event your security credentials are compromised or unauthorized access is granted.

4.0 THREAT MODEL

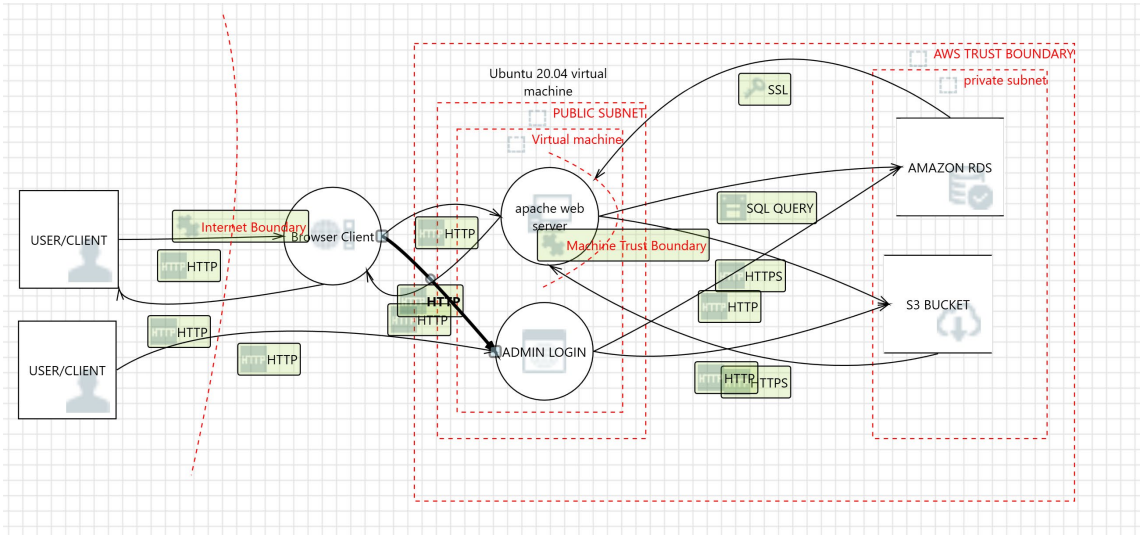


Figure 28 Screenshot of the Threat model conducted.

This is the current threat model architecture for our web application running on the AWS EC2 instance. The threat report will be attached as a separate file.

5.0 FINAL CRITICAL VULNERABILITY FINDINGS

Vulnerability	CVSS Score	OWASP Description
Apache HTTP Server 2.4.52 Vulnerability	8.8	An issue in the Apache HTTP Server allows remote attackers to execute arbitrary code or cause a denial of service (DoS) condition/HTTP Request Smuggling against backend web applications.
Cross-Site Scripting (XSS)	9.8	Allows an attacker to inject malicious scripts into web pages viewed by other users.
File Upload with No File Format Sanitization	7.5	Allows an attacker to upload malicious files without proper file format validation, potentially leading to remote code execution or other attacks.
Sensitive Directory Exposure	5.0	Occurs when sensitive directories or files are accessible to unauthorized users, potentially exposing sensitive information.
Admin Weak Credentials	8.0	Refers to weak or easily guessable credentials used by administrators, which can be exploited by attackers to gain unauthorized access to systems or sensitive information.

Figure 29 Screenshot of the CVE