Similar to the vulnerability found with SQL injection in the user's input during login, the post method for new messages took the user's input during the call to Message(). To fix this, the user's input was stored in a separate variable "user_input" which was used in the place of data.get('text'). This text was also casted to type string as an extra precaution.

```
@app.route('/message/new', methods=['POST'])
def new_message():
    token = request.headers.get('token')
    print(token)
    user = Session.query(User).filter_by(sessionToken=token).first()
    if not user:
        abort(401, 'No user.')
    data = request.get_json()
    print(request.get_json())
    user_input = str(data.get('text'))
    #if not data or not data.get('text'):
    if not data or not user_input:
        abort(409, 'No message.')
    try:
        #message = Message(user_id=user.id, content=data.get('text'))
        message = Message(user_id=user.id, content=user_input)
        Session.add(message)
        Session.commit()
    except Exception as e:
        print(e)
        Session.rollback()
    return 'Success!'
```