Jilin Zheng // U49258796

Data Sanitization/Protection Report for Project 2 and Peer-to-Peer Project

Project 2 (MongoDB)

As I am using MongoDB, I looked into NoSQL injections, using both the provided class resource, A NoSQL Injection Primer (with Mongo), and other resources I found myself, Securing MongoDB from External Injection Attacks and How to best sanitize queries? - MongoDB Developer Community Forums. The most simple solution to prevent NoSQL injections appears to be using a typed model, which I have implemented using Flask-RESTful's request parser; I am able to specify the appropriate data type of the user's input parameters. Examples of my implementation are depicted below: (my GitHub repo for reference)

```python
def init_parser(require_pass=False):
    """
    Initialize request parser.
    """
    parser = reqparse.RequestParser()
    parser.add_argument('user_name',
                        type=str,
                        location='args',
                        required=True,
                        help='You must specify a user_name.')

    if require_pass:
        parser.add_argument('user_pass',
                            type=str,
                            location='args',
                            required=True,
                            help='You must specify a password for the user.')

    return parser
```

users.py

```python
class TrainingAPI(Resource):
    def post(self):
        """
        Produces an image classification model trained on requested categories.
        """
        parser = reqparse.RequestParser()
        parser.add_argument('user_name',
                            type=str,
                            location='args',
                            required=True,
                            help='You must specify an user.')
        parser.add_argument('model_name',
                            type=str,
                            location='args',
                            required=True,
                            help='You must provide a model name.')
        parser.add_argument('categories',
                            type=str,
                            required=True,
                            location='args',
                            help='You must specify categories to train on.',
                            action='append')
        args = parser.parse_args()
```

training.py

```python
class InferenceAPI(Resource):
    def post(self):
        """
        Send image file to be inferenced with selected user's selected model.
        """
        parser = reqparse.RequestParser()
        parser.add_argument('user_name',
                            type=str,
                            location='args',
                            required=True,
                            help='You must specify an user.')
        parser.add_argument('model_name',
                            type=str,
                            location='args',
                            required=True,
                            help='You must provide a model name.')
        parser.add_argument('file',
                            type=werkzeug.datastructures.FileStorage,
                            location='files',
                            required=True,
                            help='You must provide one image!')
        parser.add_argument('inference_id',
                            type=str,
                            location='args',
                            required=True,
                            help='You must provide an inference reference id.')
        args = parser.parse_args()
```

inference.py

```python
class ImageAPI(Resource):
    """API for all image-related functions."""
    def post(self):
        """
        Save an image (file) associated with an user to ./images/{user_name}/{category}/{file.filename}.
        """
        parser = reqparse.RequestParser()
        parser.add_argument('user_name',
                            type=str,
                            location='args',
                            required=True,
                            help='You must specify an user.')
        parser.add_argument('category',
                            type=str,
                            location='args',
                            required=True,
                            help='You must provide a category for each image!',)
        parser.add_argument('file',
                            type=werkzeug.datastructures.FileStorage,
                            location='files',
                            required=True,
                            help='You must provide at least one image!')
        args = parser.parse_args()
```

image_upload.py

```
(venv) 04/26/24 18:33 DIYML $ pytest
======================== test session starts ========================
platform darwin -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: /Users/jilin/Desktop/DIYML
collected 14 items

tests/test_authentication.py ..F..                              [ 35%]
tests/test_image_upload.py ...                                  [ 57%]
tests/test_inference.py ...                                     [ 78%]
tests/test_training.py ...                                      [100%]

============================ FAILURES ============================
_____________________________ test_get_user _____________________________

    def test_get_user():
        params = {'user_name':'testName'}
        params = {'user_name':{'$ne':'null'}}
        response = requests.get(url=url,
                                params=params)
        print(response.text)
>       assert response.status_code == 200 # ok
E       assert 400 == 200
E        +  where 400 = <Response [400]>.status_code

tests/test_authentication.py:34: AssertionError
----------------------- Captured stdout call -----------------------
{
    "ERROR": "$ne does not exist!"
}

====================== short test summary info ======================
FAILED tests/test_authentication.py::test_get_user - assert 400 == 20
0
=================== 1 failed, 13 passed in 5.73s ====================
```

The pytest result shown above also shows what happens when a dictionary is passed into a field that expects a string; in this case, it appears that only the key of the input dictionary was taken as input for the MongoDB query. Also, it does not behave as an operator in MongoDB, but just as a string, as the MongoDB querying operations expect a Python dictionary containing an appropriate key-value pair, but in our case, we have only a string. Data sanitization is achieved with the strongly typed model.

Two main operators in MongoDB are also not recommended, '$where' and '$group', but this is not of concern in my code as I do not use these operators.

Peer-to-Peer Project (SQLite)

When my Peer-to-Peer system is active, the only function that accesses my SQLite database that is also accessed by users is recv_msg (image below). I have implemented a simple message sanitizer that would still enable users to see messages with single/double quotes and/or semicolons, but when the database is accessed to save messages (lines 129-132), the message *to be saved* would be stripped of single/double quotes and/or semicolons, disabling users from performing, at least, simple SQL injections. I also show the behavior of the database in [this video (5:35)]().

```
12   class Peer:
95       def recv_msg(self):
96           """
97           Handles messages sent to the initiated server
98           """
99           self.lock.acquire()
100          conn, addr = self.serv_socket.accept()
101          print(f"NEW CONNECTION ACCEPTED FROM {addr[0]}:{addr[1]}", flush=True)
102          self.lock.release()
103
104          # first received message should always be username
105          msg_header = conn.recv(self.header_len)
106          msg_len = int(msg_header.decode("utf-8").strip())
107          msg = (conn.recv(msg_len)).decode("utf-8")
108          conn_username = ""
109          if msg[0:8] == "USERNAME":
110              self.lock.acquire()
111              print(f"USERNAME RECEIVED: {msg[8:]}", flush=True)
112              self.lock.release()
113              conn_username = msg[8:]
114              # input sanitization
115              conn_username = conn_username.replace('"', "")
116              conn_username = conn_username.replace("'", "")
117              conn_username = conn_username.replace(";", "")
```

```
119          while self.session_active:
120              try:
121                  msg_header = conn.recv(self.header_len)
122                  msg_len = int(msg_header.decode("utf-8").strip())
123                  msg = (conn.recv(msg_len)).decode("utf-8")
124                  print(conn_username + " > " + msg, flush=True)
125                  # input sanitization
126                  msg = msg.replace('"', "")
127                  msg = msg.replace("'", "")
128                  msg = msg.replace(";", "")
129                  table_entry = f"""
130                      INSERT INTO recv_msgs (user, msg)
131                      VALUES (\"{conn_username}\", \"{msg}\");
132                      """
133                  self.cursor.execute(table_entry)
134                  self.db.commit()
135              except ValueError:
136                  continue
```