Массив кода

```
import pytest
import sqlite3
import sys


class AuthenticationSystem:
    def __init__(self, db_connection):
        self.connection = db_connection
        self.initialize_database()

    def initialize_database(self):
        """Инициализация таблицы пользователей в базе данных"""
        with self.connection:
            self.connection.execute(
                """
                CREATE TABLE IF NOT EXISTS users
                (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT NOT NULL UNIQUE,
                    password TEXT NOT NULL,
                    country TEXT NOT NULL,
                    balance REAL NOT NULL DEFAULT 0.0
                );
                """
            )

    def create_user(self, username, password, country, balance):
        """Создание нового пользователя с уязвимостью SQL-инъекции
(использует executescript)"""
        # УЯЗВИМОСТЬ: прямое форматирование строк позволяет SQL-инъекции,
        # и executescript позволит выполнить несколько выражений через ';'
        sql = """
        INSERT INTO users (username, password, country, balance)
        VALUES ('{}', '{}', '{}', '{}');
        """.format(username, password, country, balance)
        # executescript выполнит все выражения, включая дополнительные INSERT
из payload
        self.connection.executescript(sql)

    def verify_credentials(self, username, password):
        """Проверка учетных данных с уязвимостью SQL-инъекции"""
        cursor = self.connection.cursor()
        # УЯЗВИМОСТЬ: параметры напрямую вставляются в запрос
        cursor.execute("""
            SELECT * FROM users
            WHERE username = '{}' AND password = '{}'
        """.format(username, password))
        return cursor.fetchone()

    def remove_user(self, user_id):
        """Удаление пользователя с уязвимостью SQL-инъекции"""
        with self.connection:
            # УЯЗВИМОСТЬ: числовые параметры также уязвимы
            self.connection.execute("""
                DELETE FROM users WHERE id = {}
            """.format(user_id))

    def find_user_by_id(self, user_id):
        """Поиск пользователя по ID с уязвимостью SQL-инъекции"""
        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT * FROM users WHERE id = {}
```

```
        """.format(user_id))
        return cursor.fetchone()

    def count_users_by_country(self, country):
        """Подсчет пользователей по стране с уязвимостью SQL-инъекции"""
        cursor = self.connection.cursor()
        cursor.execute("""
            SELECT COUNT(*) FROM users WHERE country = '{}'
        """.format(country))
        return cursor.fetchone()[0]

    def transfer_funds(self, sender_id, receiver_id, amount):
        """Перевод средств между пользователями с уязвимостью SQL-инъекции"""
        with self.connection:
            # УЯЗВИМОСТЬ: все параметры подвержены инъекциям
            cursor = self.connection.cursor()
            cursor.execute("SELECT balance FROM users WHERE id =
{}".format(sender_id))
            sender_balance = cursor.fetchone()[0]

            if sender_balance < amount:
                raise ValueError("Недостаточно средств")

            self.connection.execute("""
                UPDATE users SET balance = balance - {} WHERE id = {}
            """.format(amount, sender_id))
            self.connection.execute("""
                UPDATE users SET balance = balance + {} WHERE id = {}
            """.format(amount, receiver_id))


# ФИКСТУРЫ ДЛЯ ТЕСТИРОВАНИЯ
@pytest.fixture
def database():
    """Создание временной базы данных в памяти"""
    conn = sqlite3.connect(":memory:")
    yield conn
    conn.close()


@pytest.fixture
def auth_system(database):
    """Создание системы аутентификации для тестов"""
    return AuthenticationSystem(database)


# ТЕСТОВЫЕ СЦЕНАРИИ
def test_sql_injection_in_registration(auth_system, database):
    """
    Тестирование уязвимости SQL-инъекции при регистрации.
    Демонстрация обхода ограничений через инъекцию.
    """
    # Инъекция, которая создает дополнительного пользователя
    auth_system.create_user(
        "admin', 'x', 'x', 0); INSERT INTO users (username, password,
country, balance) VALUES ('hacker','pass','test',1000); -- ",
        "password123",
        "TestCountry",
        "1000"
    )

    # Проверяем, что пользователи были созданы
    cursor = database.cursor()
    cursor.execute("SELECT username FROM users")
```

```
    users = cursor.fetchall()

    # Должны найти нескольких пользователей
    assert len(users) >= 1

    # Проверяем, что имя пользователя содержит инъекцию
    usernames = [user[0] for user in users]
    assert any("hacker" in username for username in usernames)


def test_sql_injection_in_authentication(auth_system):
    """
    Тестирование уязвимости SQL-инъекции при аутентификации.
    Обход проверки пароля через инъекцию.
    """
    # Создание тестового пользователя
    auth_system.create_user("legit_user", "correct_password", "TestCountry",
1000)

    # Обход аутентификации через SQL-инъекцию
    # Инъекция: ' OR '1'='1' -- всегда возвращает true
    user = auth_system.verify_credentials("legit_user' OR '1'='1'--",
"wrong_password")
    assert user is not None


def test_sql_injection_union_attack(auth_system, database):
    """
    Тестирование UNION-атаки через SQL-инъекцию.
    """
    # Создание тестового пользователя
    auth_system.create_user("user1", "pass1", "CountryA", 1000)

    # UNION-инъекция для получения всех пользователей
    cursor = database.cursor()
    malicious_username = "user1' UNION SELECT id, username, password,
country, balance FROM users WHERE '1'='1"

    # Эта инъекция покажет всех пользователей вместо одного
    cursor.execute(f"SELECT * FROM users WHERE username =
'{malicious_username}'")
    results = cursor.fetchall()
    assert len(results) >= 1


def test_sql_injection_always_true_condition(auth_system):
    """
    Тестирование инъекции с условием, которое всегда истинно.
    """
    auth_system.create_user("test_user", "test_pass", "TestCountry", 1000)

    # Инъекция, которая делает условие WHERE всегда истинным
    user_count = auth_system.count_users_by_country("CountryA' OR '1'='1")
    assert user_count >= 1  # Должен вернуть количество всех пользователей


def test_user_count_by_country(auth_system):
    """
    Тестирование подсчета пользователей по странам.
    """
    # Создание тестовых данных
    auth_system.create_user("user1", "pass1", "CountryA", 1000)
    auth_system.create_user("user2", "pass2", "CountryA", 1500)
    auth_system.create_user("user3", "pass3", "CountryB", 2000)
```

```
    # Проверка подсчета
    count = auth_system.count_users_by_country("CountryA")
    assert count == 2


def test_money_transfer(auth_system, database):
    """
    Тестирование функциональности перевода средств.
    """
    # Создание пользователей для перевода
    auth_system.create_user("sender", "pass1", "CountryA", 1000)
    auth_system.create_user("receiver", "pass2", "CountryB", 500)

    # Выполнение перевода
    auth_system.transfer_funds(1, 2, 300)

    # Проверка балансов
    cursor = database.cursor()
    cursor.execute("SELECT balance FROM users WHERE id = 1")
    sender_balance = cursor.fetchone()[0]
    cursor.execute("SELECT balance FROM users WHERE id = 2")
    receiver_balance = cursor.fetchone()[0]

    assert sender_balance == 700
    assert receiver_balance == 800


def test_sql_injection_in_user_deletion(auth_system):
    """
    Тестирование SQL-инъекции при удалении пользователя.
    """
    # Создаем нескольких пользователей
    auth_system.create_user("user_to_keep", "pass1", "CountryA", 1000)
    auth_system.create_user("user_to_delete", "pass2", "CountryB", 500)

    # Инъекция, которая удалит всех пользователей
    # В реальном тесте это было бы опасно, но в памяти безопасно
    try:
        auth_system.remove_user("1 OR 1=1")
    except:
        pass  # Ожидаем ошибку, так как SQLite не поддерживает множественные
операции

    # Проверяем, что пользователи удалены (демонстрация уязвимости)
    cursor = auth_system.connection.cursor()
    cursor.execute("SELECT COUNT(*) FROM users")
    count = cursor.fetchone()[0]
    assert count == 0  # инъекция "1 OR 1=1" удалила все записи


# ИНТЕРФЕЙС ВЫБОРА ТЕСТОВ
def display_menu():
    """Отображение меню выбора тестов"""
    print("=" * 60)
    print("ТЕСТИРОВАНИЕ УЯЗВИМОСТЕЙ SQL-ИНЪЕКЦИЙ")
    print("=" * 60)
    print("1. Запуск всех тестов")
    print("2. Тест SQL-инъекции при регистрации")
    print("3. Тест SQL-инъекции при аутентификации")
    print("4. Тест UNION-атаки")
    print("5. Тест инъекции с всегда истинным условием")
    print("6. Тест подсчета пользователей по стране")
    print("7. Тест перевода средств")
```

```
    print("8. Тест инъекции при удалении пользователя")
    print("9. Завершение работы")
    print("=" * 60)


def execute_tests(selection):
    """
    Запуск выбранных тестов

    Args:
        selection (str): Выбор пользователя от 1 до 9
    """
    pytest_args = [__file__, "-v"]

    test_mapping = {
        "1": "Все тесты",
        "2": ["-k", "test_sql_injection_in_registration"],
        "3": ["-k", "test_sql_injection_in_authentication"],
        "4": ["-k", "test_sql_injection_union_attack"],
        "5": ["-k", "test_sql_injection_always_true_condition"],
        "6": ["-k", "test_user_count_by_country"],
        "7": ["-k", "test_money_transfer"],
        "8": ["-k", "test_sql_injection_in_user_deletion"]
    }

    if selection == "1":
        print("Запуск всех тестовых сценариев...")
    elif selection in test_mapping:
        test_args = test_mapping[selection]
        if selection != "1":
            pytest_args.extend(test_args)
            print(f"Запуск теста: {test_args[1]}...")
    else:
        print("Неверный выбор")
        return

    result = pytest.main(pytest_args)
    return result


if __name__ == "__main__":
    """
    Основной блок программы с поддержкой двух режимов работы
    """
    if len(sys.argv) > 1:
        # Автоматический режим - запуск всех тестов
        print("Автоматический запуск всех тестов...")
        pytest.main([__file__, "-v"])
    else:
        # Интерактивный режим с меню
        print("Демонстрация уязвимостей SQL-инъекции")
        print("Внимание: система содержит преднамеренные уязвимости!")
        print("Тесты используют безопасные инъекции для демонстрации")

        while True:
            display_menu()
            user_choice = input("Введите номер теста (1-9): ").strip()

            if user_choice == "9":
                print("Завершение работы программы...")
                break
            elif user_choice in [str(i) for i in range(1, 9)]:
                execute_tests(user_choice)
                input("\nНажмите Enter для продолжения...")
```

```
            else:
                print("Ошибка: введите число от 1 до 9")
```

Меню программы

Демонстрация уязвимостей SQL-инъекции

Внимание: система содержит преднамеренные уязвимости!

Тесты используют безопасные инъекции для демонстрации

==========

ТЕСТИРОВАНИЕ УЯЗВИМОСТЕЙ SQL-ИНЪЕКЦИЙ

==========

1. Запуск всех тестов

2. Тест SQL-инъекции при регистрации

3. Тест SQL-инъекции при аутентификации

4. Тест UNION-атаки

5. Тест инъекции с всегда истинным условием

6. Тест подсчета пользователей по стране

7. Тест перевода средств

8. Тест инъекции при удалении пользователя

9. Завершение работы

Введите номер теста (1-9): 1

Запуск всех тестовых сценариев...

============================= test session starts ==============================

platform win32 -- Python 3.13.7, pytest-8.4.2, pluggy-1.6.0 --

cachedir: .pytest_cache

rootdir: C:\Users\Fall\Downloads\Telegram Desktop

collecting ... collected 7 items

main (3).py::test_sql_injection_in_registration PASSED [14%]

main (3).py::test_sql_injection_in_authentication PASSED [28%]

main (3).py::test_sql_injection_union_attack PASSED [42%]

main (3).py::test_sql_injection_always_true_condition PASSED [57%]

main (3).py::test_user_count_by_country PASSED [71%]

main (3).py::test_money_transfer PASSED [85%]

main (3).py::test_sql_injection_in_user_deletion PASSED [100%]

вывод таким образом мы познакомились с различными типами тестирования кода метод инъекции