Missão 9

Tarefa 1:

```python
import sqlite3

# Criar banco de dados e tabela
connection = sqlite3.connect(':memory:')
cursor = connection.cursor()

cursor.execute('''CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    username TEXT,
    password TEXT
)''')

cursor.execute("INSERT INTO users (username, password) VALUES ('admin', 'admin123')")
cursor.execute("INSERT INTO users (username, password) VALUES ('user', 'user123')")
connection.commit()

# Função de login insegura (sem proteção contra SQL Injection)
def login_insecure(username, password):
    query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
    cursor.execute(query)
    return cursor.fetchone()

# Teste seguro
user_safe = login_insecure("admin", "admin123")
print("Usuário encontrado (login seguro):", user_safe)

# Simulação de SQL Injection
user_injection = login_insecure("admin", "' OR '1'='1")
print("Usuário encontrado (após SQL Injection):", user_injection)

connection.close()
```

```
Usuário encontrado (login seguro): (1, 'admin', 'admin123')
Usuário encontrado (após SQL Injection): (1, 'admin', 'admin123')
```

Tarefa 2:

```python
import requests
import threading

# Função para fazer múltiplas requisições ao servidor
def send_request(url):
    while True:
        try:
            response = requests.get(url)
            print(f"Requisição enviada com status: {response.status_code}")
        except requests.exceptions.RequestException as e:
            print(f"Erro: {e}")

# URL de teste (use uma URL de um ambiente controlado)
target_url = 'http://example.com'

# Criar múltiplas threads para simular o ataque
threads = []
for i in range(100):  # Número de requisições simultâneas
    thread = threading.Thread(target=send_request, args=(target_url,))
    threads.append(thread)
    thread.start()
```

```
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200Requisição enviada com status: 200
Requisição enviada com status: 200

Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
```

Desafio 1:

1. Explique por que este código é vulnerável a SQL Injection.

Resposta: O código original era vulnerável a SQL Injection por causa da forma como as entradas do usuário eram tratadas.

2. Corrija o código para evitar a injeção SQL e faça com que a aplicação seja segura.

```python
import sqlite3

# Conectando ao banco de dados em memória
connection = sqlite3.connect(':memory:')
cursor = connection.cursor()

# Criando uma tabela e inserindo dados
cursor.execute('''CREATE TABLE users (id INTEGER PRIMARY KEY, username TEXT, password TEXT)''')
cursor.execute("INSERT INTO users (username, password) VALUES ('admin', 'admin123')")
cursor.execute("INSERT INTO users (username, password) VALUES ('user', 'user123')")
connection.commit()

# Função de login segura
def login(username, password):
    query = "SELECT * FROM users WHERE username = ? AND password = ?"
    cursor.execute(query, (username, password))
    return cursor.fetchone()

# Testando o login com SQL Injection
user = login("admin", "' OR '1'='1")
print("Usuário encontrado:", user)

connection.close()
```

```
Requisição enviada com status: 200
Requisição enviada com status: 200
Usuário encontrado: None
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200Requisição enviada com status: 200
Requisição enviada com status: 200
```

Desafio 2:

1. Explique o problema de segurança encontrado no código.

Resposta: O código é vulnerável a um ataque chamado SQL Injection, onde um invasor pode manipular a consulta SQL para obter acesso não autorizado ao sistema, para evitar isso, devemos usar **consultas parametrizadas**, que separam os dados da lógica da consulta, protegendo o banco de dados de entradas maliciosas.

2. Altere o código para proteger contra SQL Injection.

```python
import sqlite3

# Conectando ao banco de dados em memória
connection = sqlite3.connect(':memory:')
cursor = connection.cursor()

# Criando uma tabela e inserindo dados
cursor.execute('''CREATE TABLE users (id INTEGER PRIMARY KEY, username TEXT, password TEXT)''')
cursor.execute("INSERT INTO users (username, password) VALUES ('admin', 'admin123')")
cursor.execute("INSERT INTO users (username, password) VALUES ('user', 'user123')")
connection.commit()

# Função de login segura
def login(username, password):
    query = "SELECT * FROM users WHERE username = ? AND password = ?"
    cursor.execute(query, (username, password))
    return cursor.fetchone()

# Testando o login com SQL Injection
user = login("admin", "' OR '1'='1")
print("Usuário encontrado:", user)

connection.close()
```

```
Requisição enviada com status: 200Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Requisição enviada com status: 200
Usuário encontrado: None
Requisição enviada com status: 200
```