TAREFA 1

```
user_injection = login_insecure("admin", "' OR '1'='1")
print("Usuário encontrado (após SQL Injection):", user_injection)
connection.close()
```

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

TAREFA 2

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

DESAFIO 1:

Respostas:

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

O código apresentado é vulnerável a SQL Injection porque concatena diretamente os valores de username e password na consulta SQL.

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

```
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

user = login("admin", "admin123")

print("Usuário encontrado:", user)


connection.close()
```

DESAFIO 2:

1.Explique o problema de segurança encontrado no código:

O código é vulnerável a SQL Injection devido à concatenação direta do parâmetro product_name na consulta SQL. Um atacante pode manipular essa entrada para executar comandos SQL indesejados.

2.Altere o código para proteger contra SQL Injection:

```
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 products (id INTEGER PRIMARY KEY, name TEXT, price REAL)''')

cursor.execute("INSERT INTO products (name, price) VALUES ('Notebook', 2000.0)")

cursor.execute("INSERT INTO products (name, price) VALUES ('Smartphone', 1500.0)")

connection.commit()


# Função de busca de produtos segura

def search_product(product_name):
```

```
    query = "SELECT * FROM products WHERE name LIKE ?"
    cursor.execute(query, ('%' + product_name + '%',))
    return cursor.fetchall()

# Testando a busca
products = search_product("Notebook")
print("Produtos encontrados:", products)

connection.close()
```