Relatório de Análise de Segurança

SecCode Analyzer

Data: 2024-08-30

Introdução

Este relatório fornece uma análise de segurança detalhada do código fornecido, identificando vulnerabilidades potenciais e sugerindo melhorias.

Código Analisado

```
import sqlite3

# Conectar ao banco de dados SQLite
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

# Criar uma tabela de usuários
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT,
    password TEXT
)
''')

# Inserir um novo usuário (exemplo vulnerável)
def add_user(username, password):
    cursor.execute(f"INSERT INTO users (username, password) VALUES ('{username}', '{password}')")
    conn.commit()

# Autenticar usuário (exemplo vulnerável)
def authenticate(username, password):
    cursor.execute(f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'")
    user = cursor.fetchone()
    if user:
        print("Autenticado com sucesso!")
    else:
        print("Falha na autenticação!")

# Exemplo de uso
username = input("Digite seu nome de usuário: ")
password = input("Digite sua senha: ")

add_user(username, password)
authenticate(username, password)

conn.close()
```

Análise de Segurança

Analisando o código fornecido, podemos identificar algumas vulnerabilidades de segurança significativ

1. **Injeção de SQL:** As consultas SQL são construídas diretamente concatenando strings, o que torna

2. **Armazenamento de senha em texto simples:** As senhas dos usuários estão sendo armazenadas em tex

3. **Falta de validação de entrada:** Não há validação das entradas de usuário, o que pode levar a pr

Aqui estão algumas sugestões de melhorias para mitigar essas vulnerabilidades:

1. **Prevenção de Injeção de SQL:** Utilize consultas parametrizadas ao invés de concatenar strings p

```python
cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))
```

2. **Armazenamento seguro de senhas:** Em vez de armazenar senhas em texto simples, armazene hashes s

```python
import bcrypt

def add_user(username, password):
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
    cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_password
```

3. **Validação de entrada:** Sempre valide e sanitize as entradas de usuário para evitar ataques de s

```python
username = sqlite3.escape_string(input("Digite seu nome de usuário: "))
password = sqlite3.escape_string(input("Digite sua senha: "))
```

Implementando essas melhorias, você pode aumentar significativamente a segurança do seu aplicativo. I

Conclusão

A análise revelou várias áreas que podem ser melhoradas para garantir a segurança do código. As sugestões fornecidas devem ser implementadas para mitigar os riscos identificados.