```
...
def logon(username,password):
    global usu
    comando="select * from usu where logon='" + username \
        + "' and password = '" + password + "'"
    cursor = db.cursor()
    cursor.execute(comando)
    achou=False
    regs=cursor.fetchOne()
    for reg in regs:
        achou=True
        usu=reg
        if usu[5]:
            log.info(f"adm logado:{usu}")
        break
    if achou:
        return True
    return False

def alterar_prioridade(username,nova):
    if usu[5]:
        log.info(f"alterando prioridade:{usu[0]},username:{username} nova: {nova}")
        comando="update usu set prioridade=" + nova \
            + " where username='" + username + "'"
        cursor = db.cursor()
        cursor.execute(comando)
…
```

1 – O comando está concatenando argumentos SQL, o que o sujeita ao ataque de SQL Injection. Melhor seria usar a sintaxe de placeholders e argumentos;
2 – Não armazene senhas em formato clear text. Você poderia armazenar o Hash da senha no banco e passar assim para a função;
3 – Não use o mesmo mecanismo para autenticação e autorização. A mesma tabela está sendo utilizada para autenticar e para dar autorização especial ao usuário (alterar prioridade);
4 – Informações importantes estão sendo logadas em formato aberto. As senhas podem ser descobertas e adivinhadas, e dados como username e nível de acesso estão sendo enviados para fora do código e do banco de dados;
5 – Informações demais estão sendo trazidas do banco de dados. O código busca todos os dados do registro do usuário e os armazena internamente. Será que todos eles são necessários?
6 – Não há proteção contra ataque de “força bruta”. Aparentemente, o método “logon()” pode ser invocado infinitas vezes.