

Considere o seguinte Exemplo do quadro abaixo:

Para exemplificar o funcionamento da injeção de SQL, consideremos o comando básico de consulta abaixo, a instrução **SQL query**:

```
SELECT id, nome, sobrenome FROM autores WHERE nome = 'josé' AND  
sobrenome = 'silva';
```

Com base nesta instrução, é fácil supor que os itens "josé" e "silva" são do tipo texto (**strings**), solicitados por algum usuário que esteja usando a aplicação.

Problema

Portanto, supondo que a aplicação não faça o entendimento apropriado do conteúdo inserido pelo usuário, o mesmo pode fazer o uso acidental do caractere **apóstrofo**. Gerando a entrada:

- nome = *jo'sé*
- sobrenome = *silva*

E fazendo com que a aplicação gere o código:

```
SELECT id, nome, sobrenome FROM autores WHERE nome = 'jo'sé' AND  
sobrenome = 'silva';
```

De acordo com a especificação da **linguagem SQL**, o uso de apóstrofo na consulta causa uma quebra na consulta, ocorrendo um erro de **sintaxe** nessa instrução, a string será considerada no campo **nome** apenas a palavra "jo" (dentro da primeira dupla de apóstrofo 'texto'). O **interpretador** do SQL espera que a continuação da instrução sejam outros comandos SQL válidos que completem a instrução principal. No entanto, como a outra parte do texto, o "sé" não é um identificador válido, essa instrução não será executada e retornará um erro inesperado.

Ataque

Assim, um atacante pode personalizar os dados de entrada a fim de gerar um comportamento inesperado na base de dados. Para exemplificar este conceito, consideremos na consulta apresentada, a entrada dos seguintes dados através da aplicação:

- nome = *jo'; DROP TABLE autores; --*
- sobrenome = *silva*

A instrução completa ficaria:

```
SELECT id, nome, sobrenome FROM autores WHERE nome = 'jo'; DROP TABLE  
autores; --' AND sobrenome = 'silva';
```

A instrução personalizada funcionará da seguinte forma:

SELECIONE (select) todos os "ids", "nomes" e "sobrenome" DA TABELA (from) "autores" (nome da tabela) ONDE (where) os nomes deverão ser iguais a 'josé';

(quebra, novo comando) Em seguida EXCLUA (drop) a tabela "autores"; --
(continuação) E os sobrenomes iguais a 'silva' (condições do filtro);

Neste caso, a instrução será executada normalmente, pois a adição do **caractere ponto-e-vírgula ";"** na instrução representa o fim de uma SQL query e o começo de outra. Assim no exemplo acima, a SQL query será reconhecida como completa - não ocorrendo erro de sintaxe - de modo prematuro dando espaço para uma nova instrução. de livre escolha do atacante. Podendo retornar dados confidenciais armazenados na base de dados ou de executar instruções que comprometam o sistema, como a remoção de dados e/ou tabelas, como apresentado no exemplo acima.


A sequência de caracteres "--" representa um comentário em uma linha de SQL. O "--" no fim do campo username é obrigatório para que a SQL query continue sendo executada sem erros.

Pergunta 1: Explique com suas palavras qual o papel do caractere Apostrofo (') na parte grifada em amarela abaixo, do Ponto e Virgula (;) na parte grifada em amarelo e do caractere traço traço (--) parte grifada em amarelo, no exemplo dado abaixo, que representa um ataque de SQL INJECTION:

```
SELECT id, nome, sobrenome FROM autores WHERE nome = 'jo'; DROP TABLE  
autores ; --' AND sobrenome = 'silva';
```

RESPOSTA 1: O apóstrofo (') é usado na consulta SQL original para delimitar o início e o fim de uma string. No ataque, o apóstrofo fecha prematuramente a string, terminando a consulta SQL original. O ponto e vírgula (;) indica o fim de uma instrução SQL e o início de outra, permitindo ao atacante executar comandos adicionais, como DROP TABLE autores, que pode deletar uma tabela. O traço traço (--) inicia um comentário, fazendo com que o resto da consulta seja ignorado, garantindo a execução do comando DROP TABLE autores sem erros de sintaxe.

PERGUNTA 2: Considerando a tela de Login abaixo, e considerando que a mesma não tem proteção alguma contra SQL Injection. Explique como um atacante poderia se aproveitar dessa vulnerabilidade, para realizar o acesso sem saber o usuário ou a senha:



Está é a tela de login!!!

Login:

Senha:

RESPOSTA 2: Em uma tela de login sem proteção contra SQL Injection, um atacante pode inserir ' OR '1'='1 no campo de login e qualquer coisa no campo de senha. Isso transforma a consulta SQL original `SELECT * FROM usuarios WHERE login = 'usuario' AND senha = 'senha';` em `SELECT * FROM usuarios WHERE login = " OR '1'='1' AND senha = 'qualquer_coisa';`. A condição `OR '1'='1'` é sempre verdadeira, retornando todos os registros da tabela **"usuarios"** e ignorando a validação da senha. Assim, o atacante consegue acessar a aplicação sem um login ou senha válidos.