

Injeção de SQL (*SQL Injection*)

**CWE-89: Improper Neutralization of
Special Elements used in an
SQL Command ('SQL Injection')**


Injeção de SQL (*SQL Injection*)


- Agentes podem interferir nos comandos enviados ao banco de dados pela aplicação, alterando a lógica de consultas para obter dados que deveriam ser inacessíveis ou corromper o banco de dados.
- Ocorre quando acessos ao banco de dados dependem da entrada de dados pelo usuário.
- Quando a entrada de dados contém o caractere apóstrofo ('), a entrada do usuário pode ser interpretada como comando SQL, ao invés de dados do usuário.

Injeção de SQL (*SQL Injection*)

Considerar uma aplicação que requer autenticação:

Member Login





[Forgot Username / Password?](#)

```
String sql = "SELECT * FROM USUARIOS" +  
            " WHERE LOGIN = '" + usuario.getLogin() +  
            "' AND PASSWORD = '" + usuario.getPassword() + "'";  
try (Statement stmt = dbConnection.createStatement()) {  
    ResultSet rs = stmt.executeQuery(sql);  
    if (!rs.next()) {  
        throw new UserInvalidException();  
    }  
  
    User user = new User();  
    user.login = rs.getString("login");  
    user.id = rs.getString("id");  
    return user;  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```


Comando enviado
ao banco:


```
SELECT * FROM USUARIOS  
WHERE LOGIN = 'gilvanj@furb.br'  
AND PASSWORD = '1234567'
```

Injeção de SQL (*SQL Injection*)

Agora, considerar a seguinte entrada de dados

Member Login





[Forgot Username / Password?](#)

Sendo a instrução que cria o comando:

```
String sql = "SELECT * FROM USUARIOS" +  
            " WHERE LOGIN = '" + usuario.getLogin() +  
            "' AND PASSWORD = '" + usuario.getPassword() + "'";
```

Comando enviado ao banco:

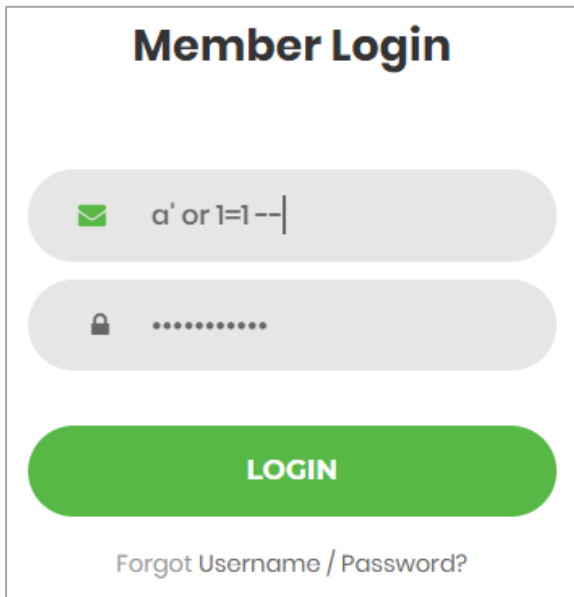
```
SELECT * FROM USUARIOS  
WHERE LOGIN = 'gilvanj@furb.br'; --'  
AND PASSWORD = '1234567'
```

Ou seja, será executado:

```
SELECT * FROM USUARIOS  
WHERE LOGIN = 'gilvanj@furb.br';
```

Injeção de SQL (*SQL Injection*)

Outra entrada possível:



Member Login

✓ a' or 1=1 --|

.....

LOGIN

[Forgot Username / Password?](#)

Sendo a instrução que envia o comando:

```
String sql = "SELECT * FROM USUARIOS" +  
            " WHERE LOGIN = '" + usuario.getLogin() +  
            "' AND PASSWORD = '" + usuario.getPassword() + "'";
```

Esta entrada cria o seguinte comando SQL:

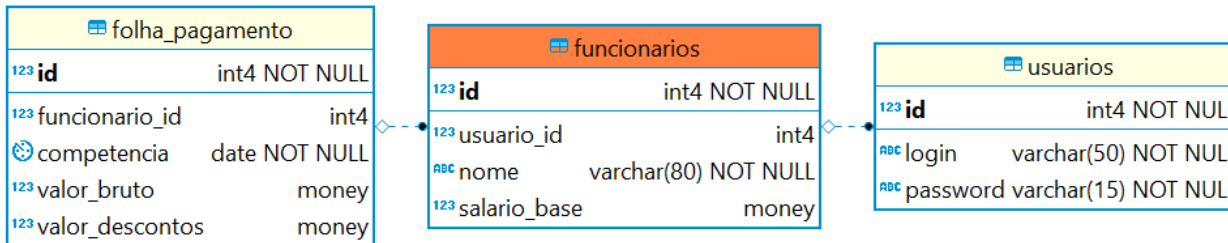
```
SELECT * FROM USUARIOS  
WHERE LOGIN = 'a' or 1=1 --'  
AND PASSWORD = '1234567'
```

Ou seja, será executado:

```
SELECT * FROM USUARIOS  
WHERE LOGIN = 'a' or 1=1
```

Injeção de SQL (*SQL Injection*)

Considerar esta modelagem:



E uma aplicação para listar as folhas de pagamento do usuário corrente:

Listagem de folha de pagamento

Competência
10/2021

LISTAR

Sendo este o comando para listar a folha de pagamento:

```
String comando = "select * from folha_pagamento" +  
    " where funcionario_id = " +  
    getCurrentUser().getFuncionario().getId() +  
    " and competencia = '01/" + data + "'";
```

Esta entrada cria o seguinte comando SQL:

```
select * from folha_pagamento where funcionario_id = 2 and  
competencia = '01/10/2021'
```

Injeção de SQL (*SQL Injection*)

Com esta entrada:

Listagem de folha de pagamento

Competência

10/2021' or 1=1 --

LISTAR

```
String comando = "select valor_bruto, valor_descontos" +  
    " from folha_pagamento" +  
    " where funcionario_id = " +  
    getCurrentUser().getFuncionario().getId() +  
    " and competencia = '01/" + data + "'";
```


Seria executado:


```
select valor_bruto, valor_descontos from folha_pagamento where funcionario_id = 2  
and competencia = '01/10/2021' or 1=1--'
```

Injeção de SQL (*SQL Injection*)

Considerar esta entrada de dados:

Member Login

 a'; DELETE FROM USUARIOS; --



LOGIN

Forgot Username / P

Esta entrada cria o seguinte comando SQL:

```
SELECT * FROM USUARIOS  
WHERE LOGIN = 'a'; DELETE FROM USUARIOS; --'  
AND PASSWORD = ''
```

Isto é, são executados estes comandos:

```
SELECT * FROM USUARIOS WHERE LOGIN = 'a';  
DELETE FROM USUARIOS;
```


Injeção de SQL (*SQL Injection*)

Numa aplicação web, se a pilha de execução for retornada para o frontend, o *agente* pode se aproveitar deste comportamento.

```
String comando = "select valor_bruto, valor_descontos" +  
    " from folha_pagamento" +  
    " where funcionario_id = " +  
    getCurrentUser().getFuncionario().getId() +  
    " and competencia = '01/" + data + "'";
```

Listagem de folha de pagamento

Competência

10/2021' having 1=1 --

LISTAR

Ops... Algo de errado aconteceu...

org.postgresql.util.PSQLException: ERROR: column "folha_pagamento.valor_bruto" must appear in the GROUP BY clause or be used in an aggregate function

Posição: 8

at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2102)

at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:1835)

at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:257)

at org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:500)

at org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statement.java:374)

Injeção de SQL (*SQL Injection*)

Listagem de folha de pagamento

Competência

10/2021' group by valor_bruto --

Foi acrescentado **group by valor_bruto**

(**having 1=1** não precisa mais manter)

Ops... Algo de errado aconteceu...

```
org.postgresql.util.PSQLException: ERROR: column "folha_pagamento.valor_descontos" must appear in the GROUP BY clause or be used in an aggregate function
```

Posição: 8

```
at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2102)
at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:1835)
at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:257)
at org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:500)
at org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statement.java:374)
at org.postgresql.jdbc2.AbstractJdbc2Statement.executeQuery(AbstractJdbc2Statement.java:254)
at usuarios.model.UsuarioDao.findRecord(UsuarioDao.java:24)
```

Injeção de SQL (*SQL Injection*)

Uma vez que se conhece a estrutura da tabela, o agente pode alterar estruturas do banco de dados:

Listagem de folha de pagamento

Competência

```
; update folha_pagamento set valor_bruto = 8888 --
```

A API em Java aplica a alteração no banco de dados, mas lança uma exceção.

```
org.postgresql.util.PSQLException: ResultSets múltiplos foram retornados pela consulta.  
at org.postgresql.jdbc2.AbstractJdbc2Statement.executeQuery(AbstractJdbc2Statement.java:258)  
at usuarios.model.UsuarioDao.findRecord(UsuarioDao.java:24)  
at autenticacao.model.Autenticacao.autenticar(Autenticacao.java:17)  
at autenticacao.controller.AutenticacaoController.authenticateUser(AutenticacaoController.java:19)  
at autenticacao.view.AutenticacaoUi.lambda$0(AutenticacaoUi.java:83)  
at java.desktop/javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1967)  
at java.desktop/javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2308)  
at java.desktop/javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:405)
```

Injeção de SQL (*SQL Injection*)

O agente pode também tentar utilizar comandos DDL (*Data definition language*) para comprometer o banco de dados.

Listagem de folha de pagamento

Competência

```
; drop table folha_pagamento; --
```

A API em Java aplica a alteração no banco de dados, embora lance uma exceção.

Injeção de SQL (*SQL Injection*)

- Um web site que permite listar dados de um pedido executa:

```
String comando = "SELECT * FROM PEDIDO WHERE ID = " +  
    request.getParameter("id");  
ResultSet rs = stmt.executeQuery(comando);
```

- Exemplo de chamada esperada:

```
http://www.website.com/pedidos?id=15362
```

- Ao chamar o web site poderia ser informado:

```
http://www.website.com/pedidos?id=15362;DROP DATABASE DB
```

- O comando que será executado é:

```
SELECT * FROM PEDIDO WHERE ID = 15362;  
DROP DATABASE DB
```

Injeção de SQL

- Exibir a pilha de execução para o usuário final pode introduzir um risco de segurança em potencial.
- Por isso, é uma boa prática de desenvolvimento de sistema seguro:

Nunca expor detalhes do erro, tal como a pilha de execução, aos usuários finais

- A pilha de execução deveria ser apresentada apenas em arquivos de log.
 - Apresentar apenas mensagens amigáveis ao usuário final
- Esta vulnerabilidade está documentada no CWE-209 (*Generation of Error Message Containing Sensitive Information*)

Injeção de SQL (*SQL Injection*)

- Os impactos com a injeção com SQL podem ser:
 - Confidencialidade – Possibilita a leitura de dados que o usuário não deveria ter acesso
 - Integridade – É possível fazer alterações ou exclusão de informações
 - Disponibilidade – Ao corromper estruturas de dados que impossibilitem aos demais usuários de utilizar a aplicação

Injeção de SQL (*SQL Injection*)

O SGBD da Microsoft possui *stored procedures* que dão acesso a funcionalidades fora do banco de dados:

- XP_CMDSHELL – Executa um comando da linha de comando
- XP_REGREAD – Lê chaves do registro do Windows
- XP_SERVICECONTROL – Acesso aos serviços

Recomendações para evitar injeção de SQL

- Limitar as entradas de dados,
 - Através de lista branca: para impedir caracteres especiais
 - Aceitar apenas os caracteres considerados válidos, recusando os demais caracteres
- O usuário de banco de dados da aplicação deve possuir privilégios reduzidos e que impeçam a execução de comandos DDL .
- Utilizar comandos SQL parametrizados.

Parametrizar comandos SQL

Acesso sem usar comando parametrizado:

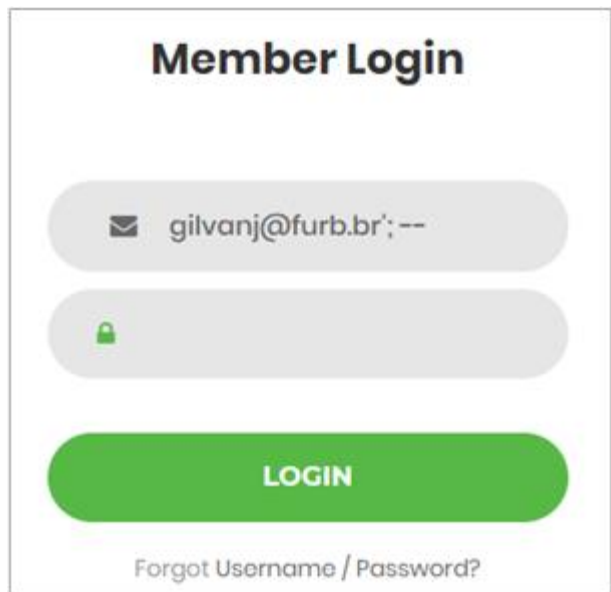
```
String sql = "SELECT * FROM USUARIOS" +  
            " WHERE LOGIN = '" + usuario.getLogin() +  
            "' AND PASSWORD = '" + usuario.getPassword() + "'";  
try (Statement stmt = dbConnection.createStatement()) {  
    ResultSet rs = stmt.executeQuery(sql);  
    if (rs.next()) {  
        return true;  
    }  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

Acesso usando comando parametrizado:

```
String sql = "SELECT * FROM USUARIOS WHERE LOGIN = ? AND PASSWORD = ?";  
try (PreparedStatement stmt = dbConnection.prepareStatement(sql)) {  
    stmt.setString(1, usuario.getLogin());  
    stmt.setString(2, usuario.getPassword());  
    ResultSet rs = stmt.executeQuery();  
    if (rs.next()) {  
        return true;  
    }  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

Parametrizar comandos SQL

Ao utilizar pesquisas parametrizadas, primeiro o comando é enviado ao banco (sem valores absolutos de parâmetros).



A login form titled "Member Login". It contains two input fields: the first for email, containing "gilvanj@furb.br'";--", and the second for password, containing a lock icon. Below the fields is a green "LOGIN" button. At the bottom, there is a link "Forgot Username / Password?".

```
"SELECT * FROM USUARIOS WHERE LOGIN = ? AND PASSWORD = ?"
```

```
gilvanj@furb.br' ;--
```

```
(string vazia)
```

Injeção de SQL (*SQL Injection*)

- Comandos que utilizam o operador *like* podem receber injeção de caracteres curingas (% , _).
- Em 2011, a prática de injeção de SQL foi responsável por comprometer sistemas de organizações como Sony Pictures, PBS, MySQL.com entre outras.