

Injeção de SQL

(SQL Injection)

Injeção de SQL (*SQL Injection*)

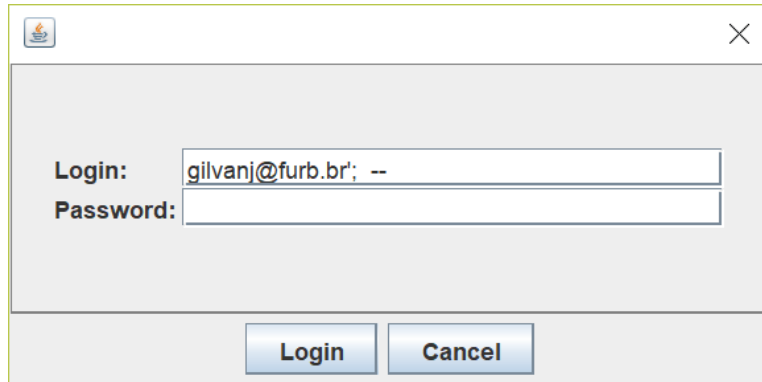
- Agentes podem interferir nos comandos enviados ao banco de dados pela aplicação, alterando a lógica de consultas para obter dados que deveriam ser inacessíveis ou corromper o banco de dados.
- Ocorre quando acessos ao banco de dados dependem da entrada de dados pelo usuário.
- Quando a entrada de dados 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 as seguintes instruções Java para validar um usuário:

```
String comando = "SELECT * FROM USUARIOS" +  
    " WHERE LOGIN = '" + usuario.getLogin() +  
    "' AND PASSWORD = '" + usuario.getPassword() + "'";  
ResultSet rs = stmt.executeQuery(comando);
```

- Considerar esta entrada de dados:



Login:

Password:

Login Cancel

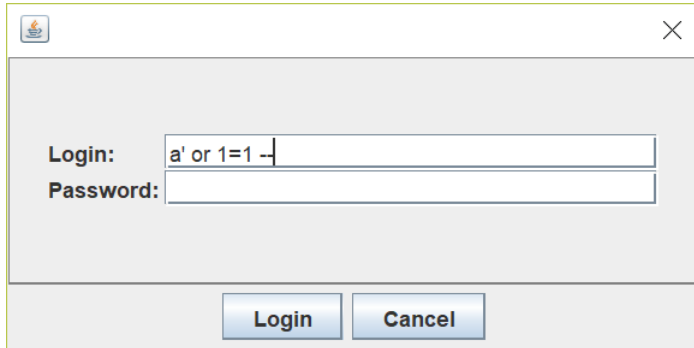
Esta entrada cria o seguinte comando SQL:

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

Injeção de SQL (*SQL Injection*)

```
String comando = "SELECT * FROM USUARIOS" +  
    " WHERE LOGIN = '" + usuario.getLogin() +  
    "' AND PASSWORD = '" +  
    usuario.getPassword() + "'";  
ResultSet rs = stmt.executeQuery(comando);
```

Considerar esta entrada de dados:



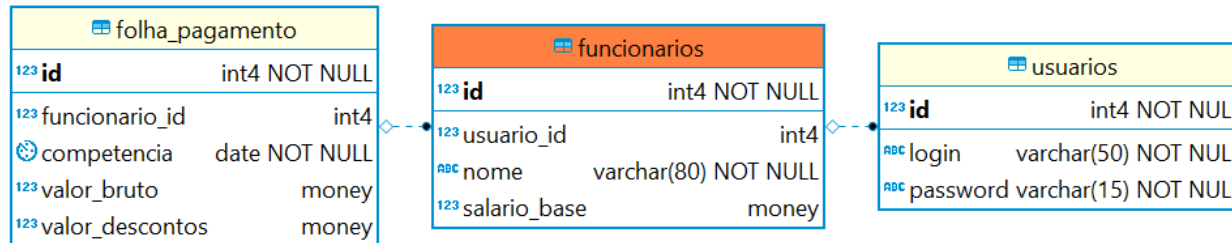
A screenshot of a web application login form. The form has two input fields: "Login:" and "Password:". The "Login:" field contains the text "a' or 1=1 --". Below the input fields are two buttons: "Login" and "Cancel". The form is enclosed in a window with a close button (X) in the top right corner.

Esta entrada cria o seguinte comando SQL:

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

Injeção de SQL (*SQL Injection*)

Considerar esta modelagem:



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

Listar folhas de pagamento

Informe o período

Data (mm/yyyy):

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 = '01/01/2018'
```

Injeção de SQL (*SQL Injection*)

Com esta entrada:

Listar folhas de pagamento

Informe o período

Data (mm/yyyy):

08/2018' or 1=1--

Enviar

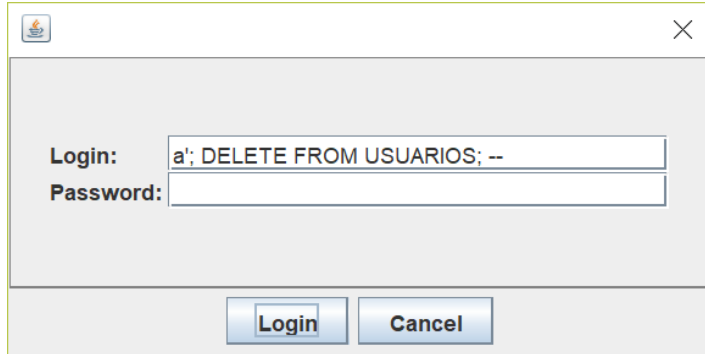
```
String comando = "select valor_bruto, valor_descontos" +  
    " from folha_pagamento" +  
    " where funcionario_id = " +  
    getCurrentUser().getFuncionario().getId() +  
    " and competencia = '01/" + data + "'";
```

Seria executado:

```
select valor_bruto, valor_descontos from folha_pagamento where funcionario_id = 2  
and competencia = '01/08/2018' or 1=1--'
```

Injeção de SQL (*SQL Injection*)

Considerar esta entrada de dados:



A screenshot of a login dialog box. It has a title bar with a small icon and a close button. The dialog contains two input fields: "Login:" and "Password:". The "Login:" field contains the text "a'; DELETE FROM USUARIOS; --". The "Password:" field is empty. At the bottom of the dialog are two buttons: "Login" and "Cancel".

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 + "'";
```

Listar folhas de pagamento

Informe o período

Data (mm/yyyy):

08/2018' having 1=1 --

Enviar

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*)

Listar folhas de pagamento

Informe o período

Data (mm/yyyy):

08/2018' group by valor_bruto having 1=1 --

Enviar

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)

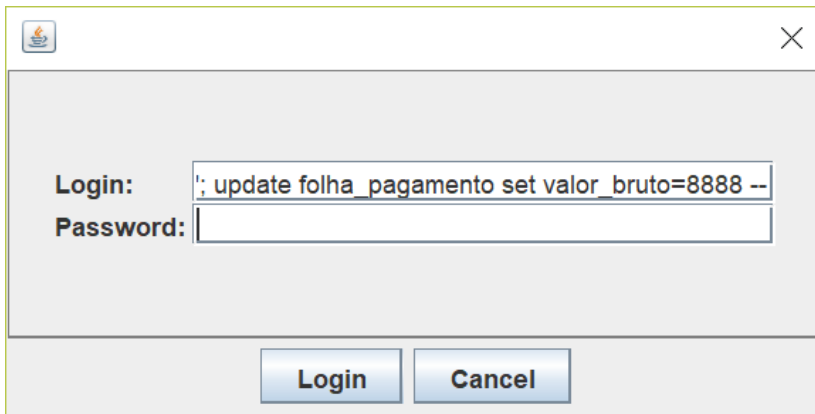
at org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statement.java:374)

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:



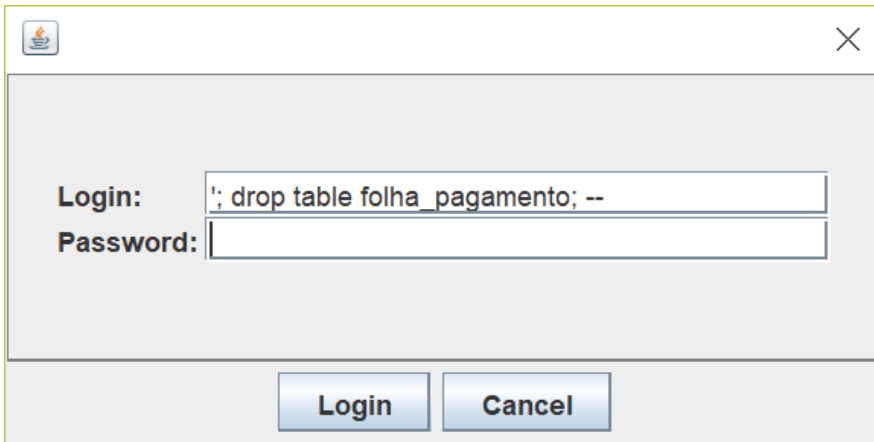
A screenshot of a Java Swing login window. The window has a title bar with a close button. Inside, there are two text input fields. The first field is labeled 'Login:' and contains the text 'update folha_pagamento set valor_bruto=8888 --'. The second field is labeled 'Password:' and is empty. Below the fields are two buttons: 'Login' and 'Cancel'.

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$0(AutenticacaoUi.java:83)  
at java.desktop/javafx.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1967)  
at java.desktop/javafx.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2308)  
at java.desktop/javafx.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.



A screenshot of a login dialog box with a title bar containing a small icon and a close button (X). The dialog has two input fields: "Login:" and "Password:". The "Login:" field contains the text `'; drop table folha_pagamento; --`. The "Password:" field is empty. At the bottom of the dialog are two buttons: "Login" and "Cancel".

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);
```

- Ao chamar o web site foi 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

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

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
 - Controle de acesso – é possível desviar do controle de proteção de autenticação
 - Integridade – É possível fazer alterações ou exclusão de informações
 - Disponibilidade – Ao corromper estruturas de dados que impossibilitem aos demais usuários de utilizar a aplicação

Recomendações para evitar injeção de SQL

- Limitar as entradas de dados, 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:

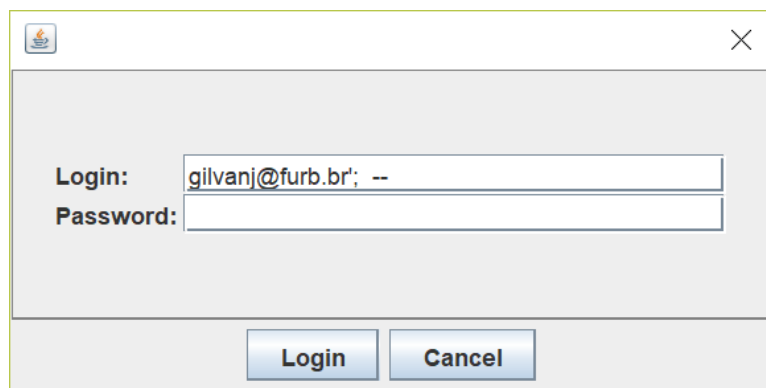
```
String sql = "SELECT * FROM USUARIOS" +  
            " WHERE LOGIN = '" + usuario.getLogin() +  
            "' AND PASSWORD = '" + usuario.getPassword() + "'";  
try (Statement stmt = dbConnection.createStatement()) {  
    ResultSet rs = stmt.executeQuery(sql);  
    if (rs.next()) {  
      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



Login:

Password:

Login Cancel

"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.