

SÉRIE WEBAPP PARA PENTESTER E APPSEC

COMO CRIAR SCANNERS

SQL INJECTION

O MANUAL PASSO A PASSO
de como criar seus próprios scripts para
identificar e tratar vulnerabilidades

FERNANDO MENGALI

SUMÁRIO

Introdução.....	01
2.0 PRÉ-REQUISITOS.....	04
3.0 CRIANDO O LABORATÓRIO/AMBIENTE.....	04
3.1 CRIANDO O BANCO DE DADOS.....	05
4.0 CRIANDO A PÁGINA PHP VULNERÁVEL.....	10
5.0 A PÁGINA PHP COM O CÓDIGO VULNERÁVEL.....	11
6.0 SITE VULNERÁVEL.....	13
7.0 TEORIA: COMO DETECTAR SQL INJECTION.....	14
8.0 PRÁTICA: DETECTANDO INJEÇÃO DE SQL.....	15
9.0 CONSTRUÇÃO DO SCANNING.....	15
10.0 PERL NO LINUX.....	18
11.0 CODIFICANDO A FERRAMENTA DE AUTOMAÇÃO.....	18
12.0 IMPLEMENTAÇÕES.....	24
13.0 CÓDIGO COMPLETO.....	27
14.0 CORRIGINDO VULNERABILIDADE.....	31
15.0 PROTEÇÃO COM WAF MODSECURITY OU NAXSI.....	33
16.0 SOBRE O AUTOR.....	34

INTRODUÇÃO

Nesse artigo, desenvolveremos uma ferramenta com a linguagem de programação Perl que identificará páginas de internet que possuem vulnerabilidades de Injeção de SQL.

Primeiro, iremos apresentar o processo de **identificação manual da vulnerabilidade de SQL Injection**, posteriormente você aprenderá como desenvolver um **script em Perl para detectar automaticamente** esse tipo de vulnerabilidade.

Esse artigo não apresenta técnicas avançadas para o desenvolvimento do nosso script em Perl para a identificação de vulnerabilidades. Para a elaboração desse artigo, utilizamos conceitos básicos, mas eficiente para identificar vulnerabilidades de SQL Injection, seja para um alvo específico ou vários alvos.

O conteúdo sobre como identificar vulnerabilidades de SQL Injection nesse artigo não são equivalentes as grandes ferramentas de mercado que atendem a metodologia DAST (Dynamic application security testing).

Não ensinamos a desenvolver algoritmos sofisticados que são utilizadas pelas ferramentas de análise dinâmica disponíveis comercialmente, mas compartilhamos informações suficientes para começar a criar suas primeiras ferramentas para identificar vulnerabilidades e continuar aperfeiçoando suas técnicas de desenvolvimento de scripts de identificação de vulnerabilidades.

2.0 PRÉ-REQUISITOS

Será necessário instalar os softwares abaixo para o desenvolvimento do laboratório:

- Sistema operacional **Microsoft Windows** (no artigo utilizei o Windows 10)
- Download do **WAMP 3.1.9**:
<https://sourceforge.net/projects/wampserver/>
- Download **Perl**:
<https://www.activestate.com/products/activeperl/downloads/>

3.0 CRIANDO O LABORATÓRIO/AMBIENTE

Nessa seção instalaremos o WAMP (Apache, MySQL e PHP) no Windows. Até o desenvolvimento desse artigo, foi utilizado o **WAMP 3.1.9**.

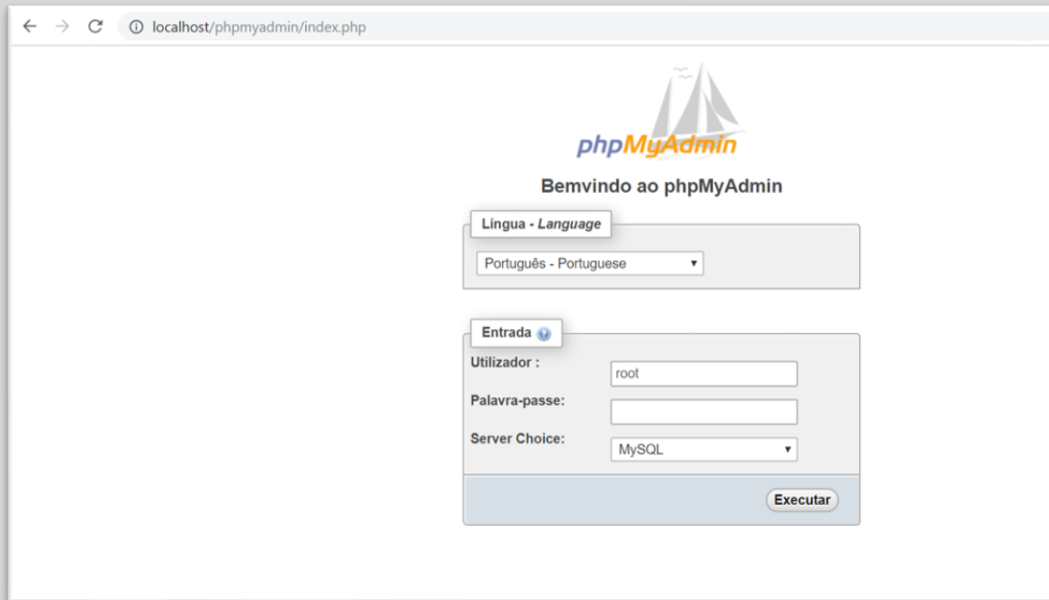
O processo de instalação é muito simples, portanto, não abordaremos.

Vamos considerar que você concluiu a instalação do WAMP e depois de instalado, vamos prosseguir com as configurações.

Se desejar acessar somente a seção sobre o desenvolvimento do scanning em Perl, acesse a **seção 8**.

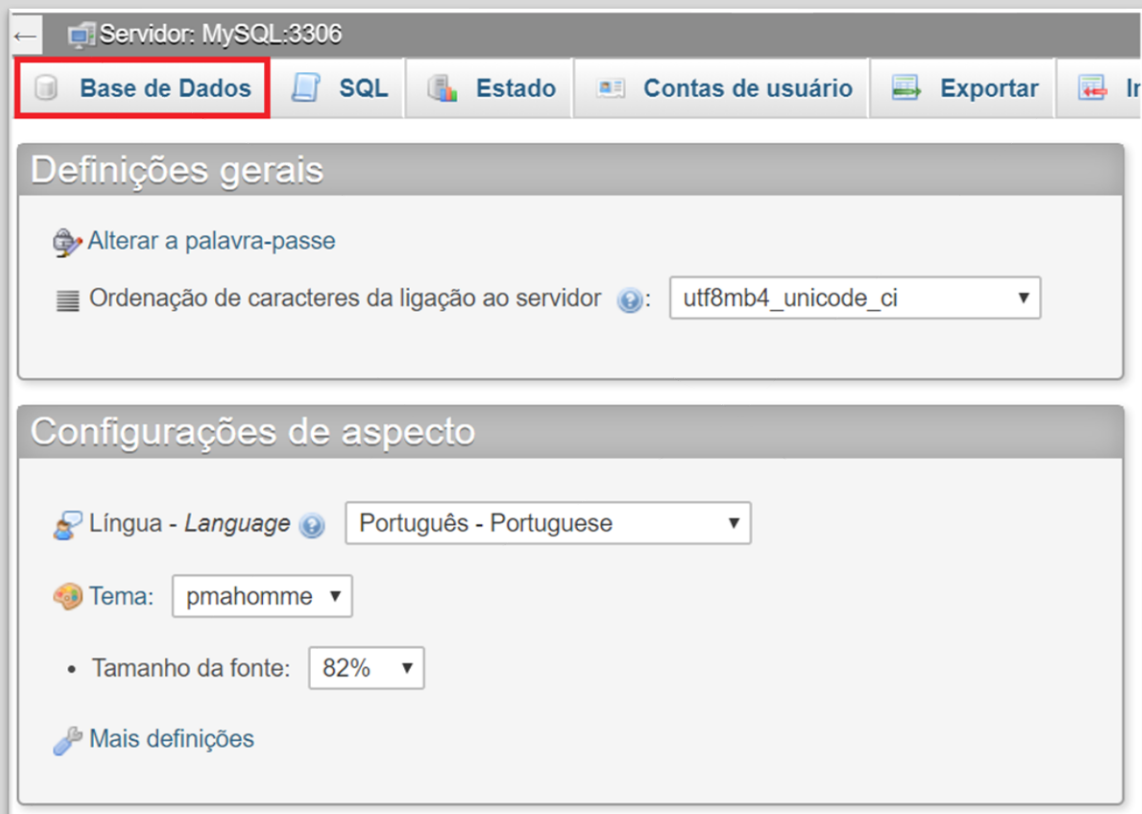
3.1 CRIANDO O BANCO DE DADOS

Acesse o phpMyAdmin: <http://localhost/phpmyadmin/index.php>

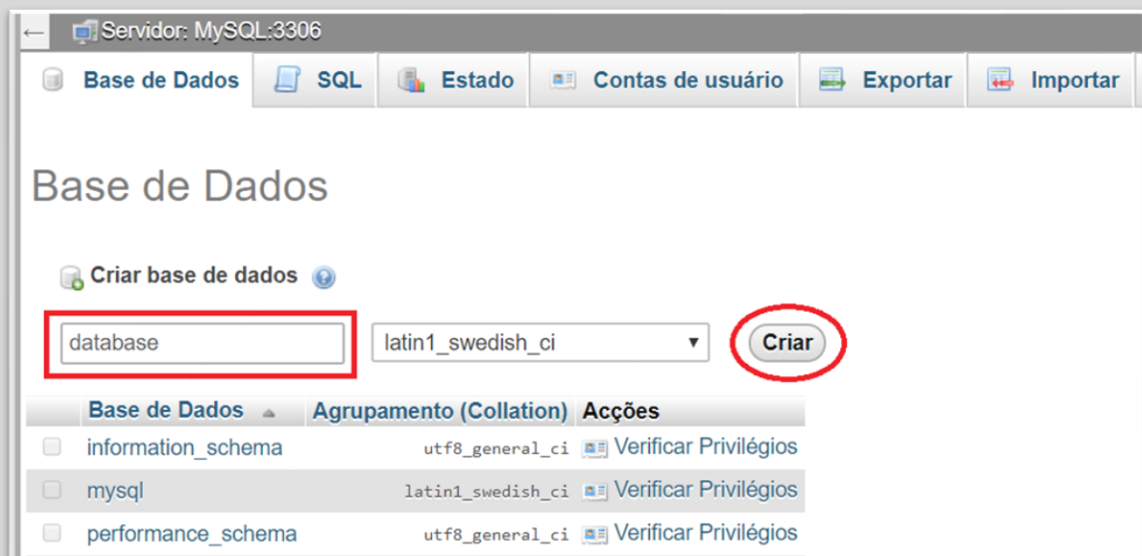


The screenshot shows the phpMyAdmin login interface in a web browser. The browser's address bar displays 'localhost/phpmyadmin/index.php'. The page features the phpMyAdmin logo at the top, followed by the text 'Bemvindo ao phpMyAdmin'. Below this, there is a language selection dropdown menu labeled 'Lingua - Language' with 'Português - Portuguese' selected. Underneath is an 'Entrada' (Login) section containing three input fields: 'Utilizador :' (Username) with 'root' entered, 'Palavra-passe:' (Password) which is empty, and 'Server Choice:' with 'MySQL' selected from a dropdown. An 'Executar' (Execute) button is located at the bottom right of the login form.

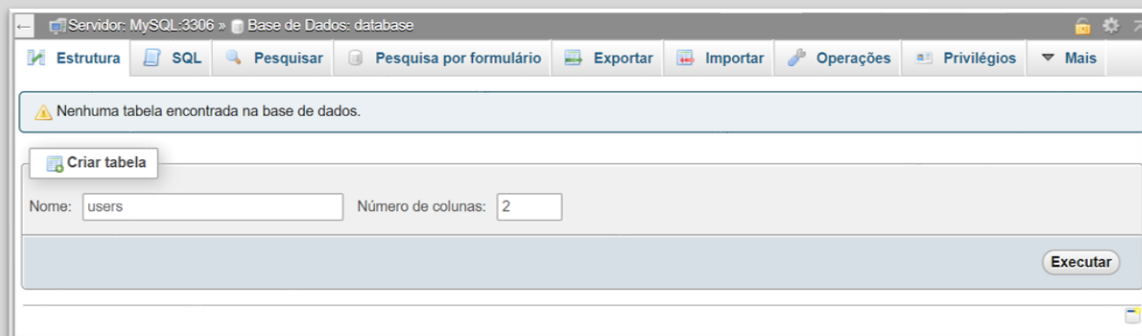
3.1.1 Por padrão, o phpMyAdmin tem o nome de usuário **root** e sem senha.



3.1.2 A próxima etapa, será criar o banco de dados.
Clique na aba **Base de Dados** para criar o banco de dados.



3.1.3 Digite o nome do banco de dados e depois clique em **Criar**.
No nosso exemplo, o nome do banco de dados é **"database"**.



3.1.4 Depois de criado o banco de dados **database**, vamos criar a **tabela**.

Acesse sua base de dados e depois a aba “**Estrutura**”.

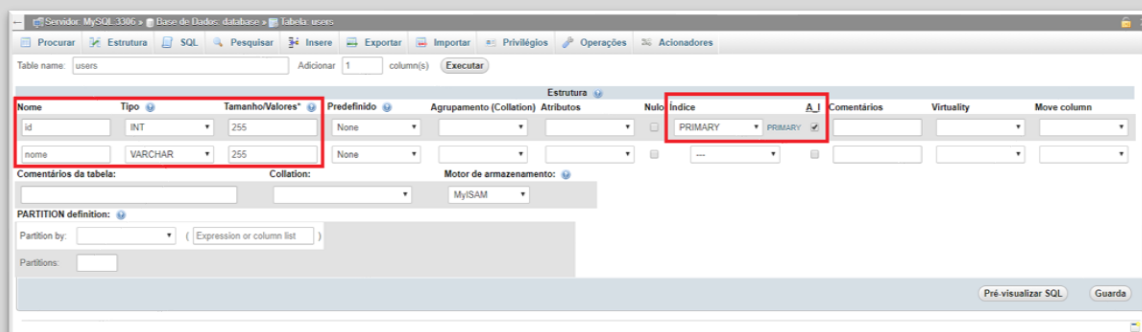
No campo **Nome**, digite o nome da tabela.

No nosso exemplo a tabela terá o nome de “**users**”.

No campo **Número** de colunas, digite a quantidade de colunas.

No nosso exemplo será 2 colunas.

Depois clique em **Executar**.



3.1.5 Na coluna **Nome**, informe o nome das colunas.

No exemplo será **id** e **nome**.

Na coluna **Tipo** defina o campo **id** como **INT** e o tamanho de **255**.

No campo **nome**, defina o **Tipo** como **VARCHAR** e o **Tamanho** **255**.

Não esqueça da coluna **índice**, será necessário definir como **PRIMARY**.

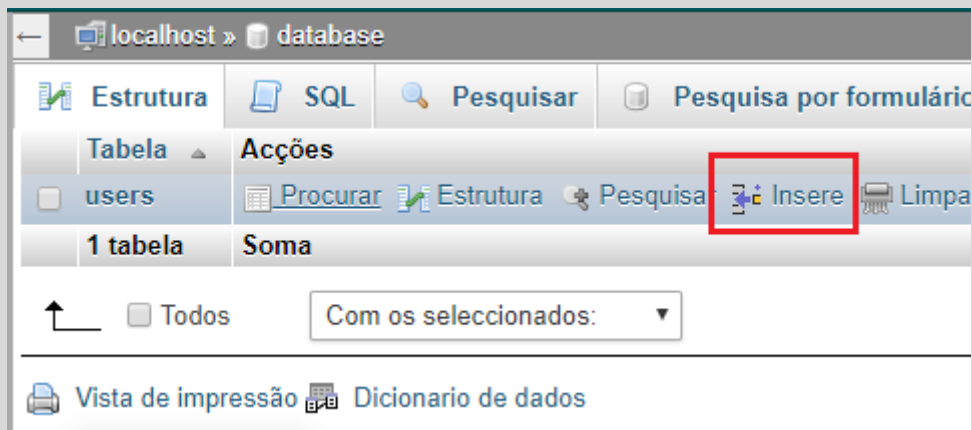
Na coluna **A_I**, defina como **PRIMARY** para colunas do **id**.

Caso deseje ser mais rápido, acesse a segunda aba chamada de **SQL** e informe o código abaixo para criar sua tabela:

```
CREATE TABLE IF NOT EXISTS `users` (
  `id` int(255) NOT NULL AUTO_INCREMENT,
  `nome` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
COMMIT;
```



3.1.6 Depois clique em “Executar”.



3.1.7 Agora, vamos criar um usuário.

Clique na aba “Estrutura” e depois clique no link “Inserir”.

Coluna	Tipo	Funções	Nulo	Valor
id	int(255)			
nome	varchar(255)			John Doe

Executar

3.1.8 Adicione o usuário **John Doe** e depois clique na opção “**Executar**”.

Coluna	Tipo	Funções	Nulo	Valor
id	int(255)			
nome	varchar(255)			John Doe

Executar

✓ 1 linha inserida.
Inserted row id: 1

```

INSERT INTO `users` (
  `id`,
  `nome`
)
VALUES (
  NULL, 'John Doe'
);

```

[Editar] [Criar código PHP]

Tabela	Registos	Tipo	Agrupamento (Collation)	Tamanho	Suspensão
users	1	MyISAM	latin1_swedish_ci	2 KB	-
1 tabela	Soma	1	InnoDB	latin1_swedish_ci	2 KB 0 Bytes

↑ ☐ Todos Com os seleccionados: ▼

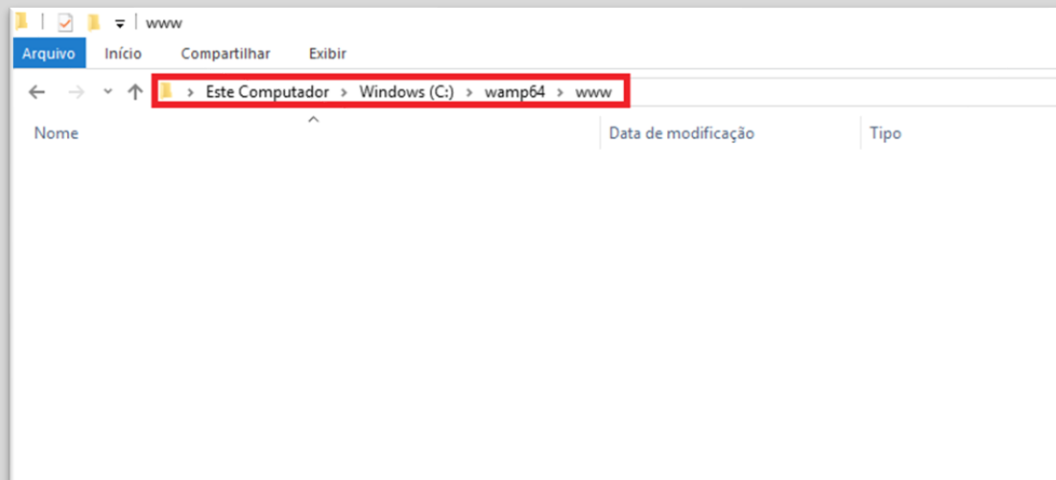
3.1.9 O usuário foi criado com sucesso.

Pronto, concluímos todas as etapas necessárias do banco de dados!
Agora, vamos para a última etapa, o desenvolvimento da página vulnerável.

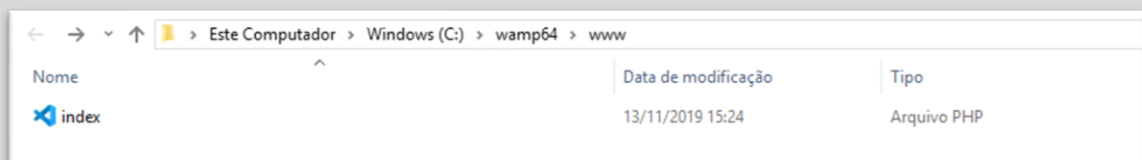
4.0 CRIANDO A PÁGINA PHP VULNERÁVEL

Acesse o diretório **www** para criarmos a página em PHP. Se você utilizou a sugestão do Windows para a instalação, o caminho será

“C:\Windows\wamp64\www”. Veja abaixo:



4.1.1 Quando você acessar o conteúdo do diretório **“www”**, visualizará alguns arquivos. Particularmente, eu removi todos os arquivos, deixando o diretório **“www”** vazio. A remoção dos arquivos do diretório **“www”** é sua escolha, eu acho melhor para trabalhar.



4.1.2 Nessa etapa iremos criar um arquivo com a extensão **“PHP”** com o nome de **“index”** no diretório **“www”**.

Depois de criar a página index, iremos adicionar o conteúdo ou código PHP vulnerável na página **index.php**.

Se você não codifica em PHP, não se preocupe, abaixo apresentamos o código e depois descrevemos o funcionamento de cada linha.

5.0 A PÁGINA PHP COM O CÓDIGO VULNERÁVEL

Abaixo apresentamos o código PHP vulnerável completo:

```
<?php

if (isset($_GET['id'])){
    $id = $_GET['id'];

    /* Setup the connection to the database */
    $mysqli = new mysqli('localhost', 'root', '', 'database');

    /* Check connection before executing the SQL query */
    if ($mysqli->connect_errno) {
        printf("Connect failed: %s\n", $mysqli->connect_error);
        exit();
    }

    /* SQL query vulnerable to SQL injection */
    $sql = "SELECT username FROM users WHERE id = '$id'";

    /* Select queries return a result */
    if ($result = $mysqli->query($sql)) {

        while($obj = $result->fetch_object()){
            print($obj->username);
        }
    }

    /* If the database returns an error, print it to screen */
    elseif($mysqli->error){
        print($mysqli->error);
    }
    else {

    }

}

?>
```

5.0.1 Você poderá copiar esse código e adicionar para a sua página **index.php**.

Não esqueça, sua página **index.php** deverá estar em “C:\Windows\wamp64\www”.

Essa etapa é bem simples, você não precisa ter conhecimentos de PHP para entender o código.

Se você quiser entender o código, continue lendo essa seção, pois descreverei cada linha na próxima página.

Inicialmente, nosso código receberá um parâmetro GET:

```
if (isset($_GET['id'])){  
    $id = $_GET['id'];
```

5.0.2 Temos o if que valida a existência de dados ou parâmetros enviados para o método GET. Se houver algum dado trafegando via GET ele entrará no commando bloco IF e será armazenado na variável **id**.

Vejo o exemplo de acesso a nossa página PHP via a url: <http://localhost/index.php?id=1>

Após recebermos o valor, iniciamos a conexão com o banco de dados “**database**”.

Nas próximas linhas, **validamos** se a conexão com o banco de dados está funcionando.

```
$mysqli = new mysqli('localhost', 'root', '', 'database');
```

5.0.3 Nessa linha ocorre a conexão com o banco de dados.

A próxima etapa é validar a conexão com o banco de dados.

```
if ($mysqli->connect_errno) {  
  
    printf("Connect failed: %s\n", $mysqli->connect_error);  
    exit();  
  
}
```

5.0.4 No segundo if, validamos se há problemas na conexão com o banco de dados.

Se a conexão funcionar, seguimos para a seção **5.0.5** que apresenta a consulta com o banco de dados.

Senão funcionar, tivemos um problema na conexão com o banco de dados e receberemos uma mensagem de erro.

A mensagem de erro sempre ajudará na identificação do problema na conexão com o banco de dados.

```
/* SQL query vulnerable to SQL injection */  
$sql = "SELECT username FROM users WHERE id = '$id'";
```

5.0.5 Nessa linha, iniciamos a consulta com o nome de usuário na tabela **users**. A consulta na tabela **username** utilizará o campo **id** e o valor armazenado na variável **\$id** para criar uma condição na busca.

```
/* Select queries return a result */  
if ($result = $mysqli->query($sql)) {  
    while($obj = $result->fetch_object()){  
        print($obj->username);  
    }  
}
```

5.0.6 Agora, utilizamos a query armazenada na variável **\$sql** no **while**. Depois, utilizamos o **fetch_object** que retorna os dados como um objeto. O comando **print**, será responsável por imprimir ou apresentar os dados armazenados no banco de dados.

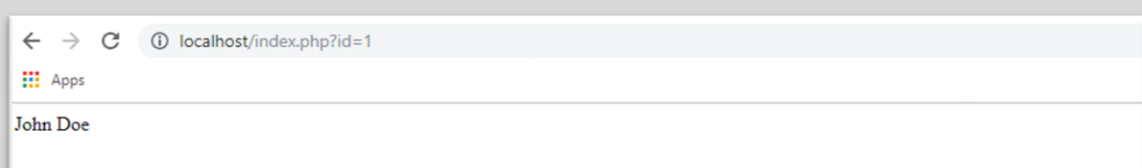
Se a nossa query gerar algum erro, será exibido na tela as informações do erro.

6.0 SITE VULNERÁVEL

Nessa seção acessaremos a página **index.php** vulnerável:

<http://localhost/index.php?id=1>.

Veja o resultado no browser:



6.0.1 O nome **John Doe** é exibido no browser, pois nome está associado ao número 1 da tabela **users** do banco de dados.

7.0 TEORIA: COMO DETECTAR INJEÇÃO DE SQL

Para identificar o nosso alvo vulnerável, utilizaremos uma técnica simples: **as aspas simples**.

Na nossa URL de exemplo, a página **index.php** possui uma chave ou índice denominado **id** e um respectivo valor, o número **1**.

O número **1** está associado ao nome do usuário **John Doe** registrado na tabela **users** do banco de dados.

Entendido o funcionamento para a exibição de conteúdo por meio de parâmetros com o número de **id**, vamos entender como funciona a detecção de SQL Injection.

Quando um atacante deseja encontrar uma vulnerabilidade de SQL Injection será preciso adicionar uma aspa simples no final da URL:

<http://localhost/page.php?id=1'>

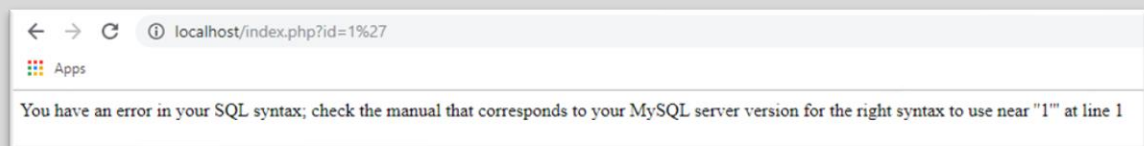
Se a aplicação possuir alguma tipo de vulnerabilidade de Injeção de SQL, os dados não serão exibidos e uma mensagem de erro, parecida com o resultado abaixo será apresentada no browser:

```
you have an error in your sql syntax check the manual that corresponds to  
your mysql server version for the right syntax to use near ''' at line
```

A mensagem acima, significa que a aplicação possui uma falha na codificação e está vulnerável à injeção de SQL.

8.0 PRÁTICA: DETECTANDO A VULNERABILIDADE DE INJEÇÃO DE SQL

Agora vamos descobrir se a página está vulnerável a Injeção de SQL. Para essa etapa, utilizaremos uma aspa simples: " ' ".Vejam os:



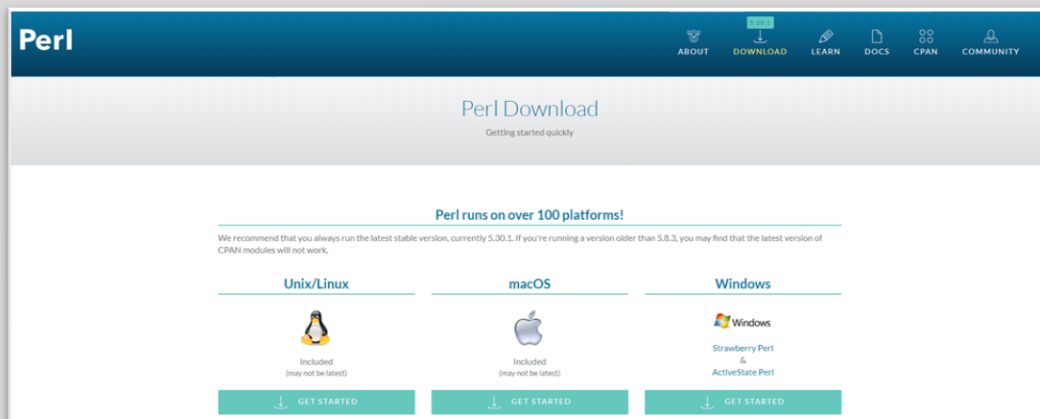
8.0.1 A página está vulnerável a injeção de SQL.

Entendemos como funciona a vulnerabilidade de Injeção de SQL, agora desenvolveremos a ferramenta para a automatização de identificação de vulnerabilidade Injeção de SQL, mas antes vamos instalar o interpretador de Perl para programar o script ou ferramenta em Perl.

9.0 CONSTRUÇÃO DO SCANNING

A linguagem de desenvolvimento escolhida para o desenvolvimento do script será o Perl. Você precisará de conhecimentos de programação em Perl, pois a ferramenta terá erros propositalmente, ou seja, apenas desenvolvedores, analistas de segurança e interessados com aptidões de desenvolvimento entenderão o código.

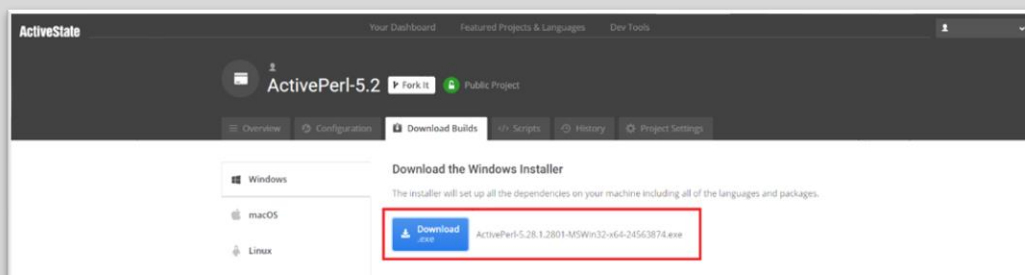
9.1 BAIXANDO O PERL PARA WINDOWS



9.1.1 Acesse a URL <https://www.perl.org/get.html> e escolha a plataforma que você está utilizando.

Você será redirecionado e solicitado a autenticar ou criar uma conta para baixar o Perl.

Depois de autenticado, você poderá baixar o Perl:



9.1.2 Clique no botão “Download”.

9.2 OPÇÃO 2: STRAWBERRY PERL PARA WINDOWS

Outra opção é utilizar **Strawberry Perl**:

<http://strawberryperl.com/releases.html>

Strawberry Perl

[back to homepage](#)

Explanatory Notes

- MSI installer** - preferred way, requires admin privileges to install
- ZIP edition** - admin privileges not required, however you need to run some initial scripts manually after unzip
- Portable edition** - suitable for "perl on USB stick" (you can move/rename the perl directory and it will still work)
- PDL edition** - portable edition + extra PDL related modules and external libraries

Recommended downloads

version	date	MSI edition	Perl64	Perl edition	ZIP edition
5.30.0.1	2019-05-23	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.2.1	2019-09-02	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.3.1	2019-11-02	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.4.1	2019-04-15	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.5.1	2017-01-15	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.6.1	2016-07-09	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.4.1	2014-10-02	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.3.1	2013-03-13	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit
5.30.4.1	2013-03-13	32bit / 64bit	32bit / 64bit	32bit / 64bit	32bit / 64bit

Strawberry Perl 5.30.0.1 (2019-05-23)

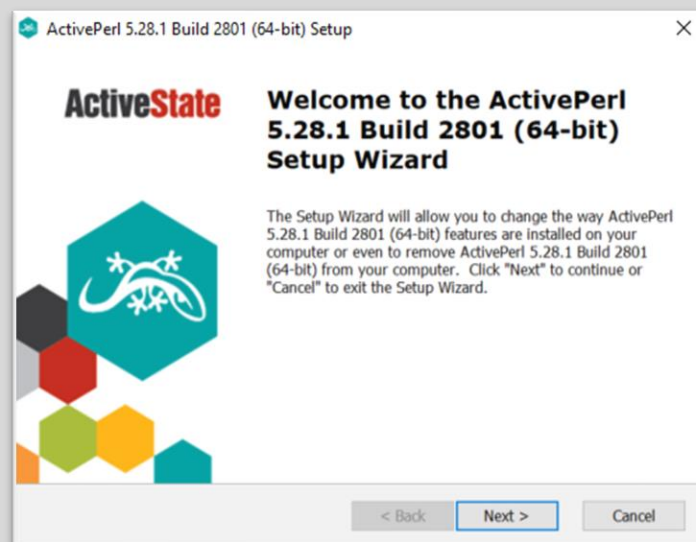
- May 2019 / 5.30.0.1 / 64bit - [Release Notes](#)

download	SHA1 digest	Size
MSI installer	21a79f4f8f8b3dc3b1d5d1a3bdc7404720a5	187.1 MB
Perl edition	96c9f7f2336e299fa7b4d037606f1a477047	178.9 MB
Portable edition	14ca71a5123f4a6b4ebc9f765c1d34477957	155.0 MB
ZIP edition	9b9d120b0b0f4f2ff0f2621c510e42f7957	155.0 MB

- May 2019 / 5.30.0.1 / 32bit / with USE_64_BIT_INT - [Release Notes](#)

9.2.1 Veja a página acima.

Depois de baixar o Perl para Windows, siga os procedimentos comum de instalação no Windows:



9.2.2 O restante você já sabe.

10.0 PERL NO LINUX

Se você está utilizando uma distribuição Linux, de preferência o Kali Linux, por padrão o Perl já está instalado.

Caso você deseje verificar se o Perl está instalado, digite os comandos **perl** **-help** no terminal do Kali Linux:

```
root@kali:~# perl -help
Usage: perl [switches] [--] [programfile] [arguments]
  -0[octal]          specify record separator (\0, if no argument)
  -a                autosplit mode with -n or -p (splits $_ into @F)
  -C[number/list]    enables the listed Unicode features
  -c                check syntax only (runs BEGIN and CHECK blocks)
  -d[:debugger]      run program under debugger
  -D[number/list]    set debugging flags (argument is a bit mask or alphabets)
  -e program         one line of program (several -e's allowed, omit programfile)
  -E program         like -e, but enables all optional features
  -f                don't do $sitelib/sitecustomize.pl at startup
  -F[pattern/]       split() pattern for -a switch (//s are optional)
  -i[extension]      edit <> files in place (makes backup if extension supplied)
  -Idirectory        specify @INC/#include directory (several -I's allowed)
  -l[octal]          enable line ending processing, specifies line terminator
  -[mM][...]module   execute "use/no module..." before executing program
  -n                assume "while (<>) { ... }" loop around program
  -p                assume loop like -n but print line also, like sed
  -s                enable rudimentary parsing for switches after programfile
  -S                look for programfile using PATH environment variable
  -t                enable tainting warnings
  -T                enable tainting checks
  -u                dump core after parsing program
  -U                allow unsafe operations
  -v                print version, patchlevel and license
  -V[:variable]      print configuration summary (or a single Config.pm variable)
  -w                enable many useful warnings
  -W                enable all warnings
  -x[directory]      ignore text before #!perl line (optionally cd to directory)
  -X                disable all warnings

Run 'perldoc perl' for more help with Perl.

root@kali:~#
```

11.0 CODIFICANDO A FERRAMENTA DE AUTOMAÇÃO

Nessa etapa iremos utilizar uma requisição para nossa página <http://localhost/index.php?id=1>.

Utilizaremos umas aspas simples e trataremos a resposta.

11.1 CLASSES DE REQUISIÇÕES

Nosso código precisará de duas classes para fazer requisições para a página com vulnerabilidade.

São elas:

- LWP::UserAgent
- HTTP::Request
- LWP::Simple

LWP::UserAgent

É uma classe responsável por atuar como um agente, durante uma requisição ou solicitação da web. Quando uma requisição é realizada será criado um objeto LWP::UserAgent com valores padrões.

HTTP::Request

A classe HTTP::Request faz uma requisição da URL ou página web que definiremos.

Conforme apresentado acima, teremos o cabeçalho **HTTP::Request** no nosso código para automatizar requisições.

LWP::Simple

É uma versão simplificada da biblioteca libwww-perl.

Possui várias funções e possibilita maior controle nos campos de cabeçalho.

O LWP::Simple busca rapidamente uma página e devolve a resposta. As respostas poderão ser: **is_error** ou **is_success**.

11.2 COMEÇANDO COM A CODIFICAÇÃO

Utilizando os três módulos descritos acima, poderemos adicioná-los no início do script e depois criar a estrutura de requisição em Perl para a página vulnerável.

```
1  #!/usr/bin/perl
2
3  use LWP::UserAgent;
4  use HTTP::Request;
5  use LWP::Simple;
6
```

11.2.1 Não esqueça de usar `#!/usr/bin/perl`, se estiver usando linux.

11.3 ENTRADA DE DADOS

A entrada de dados será utilizada para informar qual URL será verificada. Caso não queira informar a URL utilizando uma entrada de dados, poderá deixar o endereço de forma estática na variável.

Na **linha 7** criamos uma mensagem ou um prompt para o usuário digitar a URL.

Na **linha 8** criamos uma variável `$url` e depois adicionamos a entrada padrão `<stdin>`.

STDIN, poderá ser substituída por `<>`.

```
6
7  print "Por favor, informe a url: \n";
8  $url = <stdin>;
9
```

11.3.1 Caso o desenvolvedor não opte por utilizar uma entrada de dados, poderá utilizar uma variável estática para armazenar a URL que será verificada, exemplo:

```
9
10 $url = "http://localhost/index.php?id=1";
11
```

11.3.2 Aproveitando a variável `$url`, vamos adicionar uma aspa simples ao final da url.

```
11
12  $url = $url." ";
13
```

11.3.3 No exemplo acima, estamos armazenando o endereço da url mais a aspa simples na variável \$url.

Agora, podemos desenvolver a arquitetura da requisição:

```
13
14  my $requisicao=HTTP::Request->new(GET=>$url);
15
16  my $ua=LWP::UserAgent->new();
17  exit;
18  $ua->timeout(15);
19
20  my $resultado=$ua->request($requisicao);
21
```

11.3.4 A estrutura da requisição.

“Nessa etapa, exige conhecimentos de programação em Perl ou similar”.

Na **linha 14** utilizamos o módulo HTTP::Request, o método GET e o endereço da URL que analisaremos a resposta.

Na **linha 16** utilizamos um UserAgent e na **linha 18** passamos um parâmetro de 15 segundos de timeout.

Na **linha 20** a variável \$resultado armazena a requisição que será executada.

Como nossa requisição acessa uma página com uma aspa simples no final da URL, teremos como resposta o erro.

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "1" at line 1

Nessa etapa recebemos a resposta do conteúdo da página. Se o conteúdo contém ***You have an error in your SQL syntax*** a página é vulnerável.

```

21
22  if ($resultado->content =~ /You have an error in your SQL syntax/) {
23
24      print "Vulnerável!";
25
26  }
27  else {
28      print "Não vulnerável!";
29  }
30

```

11.3.5 Podemos adicionar outros erros para serem comparados com o conteúdo de uma página.

11.4 ERROS DE BANCO DE DADOS

MYSQL

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "1" at line 1

MICROSOFT SQL SERVER COM ASPX

Server Error in '/' Application

ORACLE

ORA-00933: SQL command not properly ended

POSTGRESQL

PSQLException: ERROR: unterminated quoted string at or near ""

Query failed: ERROR: syntax error at or near ""

MICROSFT SQL SERVER

Microsoft SQL Native Client error '80040e14' Unclosed quotation mark after the character string

Microsoft SQL Native Client error %u201880040e14%u2019 Unclosed quotation mark after the character string

[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value

ERROS DE ODBC

Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

MICROSOFT ACCESS COM ASP (SERVIDOR IIS)

Microsoft JET Database Engine error '80040e14'

MICROSFT ACCESS COM PHP (SERVIDOR APACHE)

Fatal error: Uncaught exception 'com_exception' with message Source: Microsoft JET Database Engine

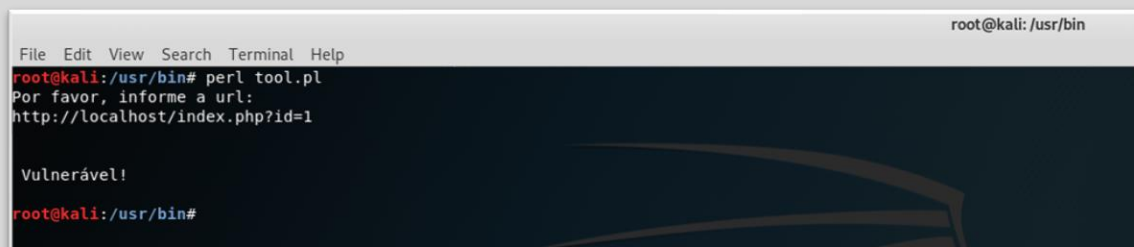
Adicionando mais mensagens de erros de banco de dados ao nosso script.

```
21
22     if ($resultado->content =~ /You have an error in your SQL syntax/
23         || $resultado->content =~ /Server Error in/
24         || $resultado->content =~ /SQL command not properly ended/
25         || $resultado->content =~ /PDOException: ERROR: unterminated quoted string at or near/
26         || $resultado->content =~ /Query failed: ERROR: syntax error at or near/
27         || $resultado->content =~ /Microsoft SQL Native Client error/
28         || $resultado->content =~ /[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value/
29         || $resultado->content =~ /Microsoft OLE DB Provider for ODBC Drivers/
30         || $resultado->content =~ /Microsoft JET Database Engine error/
31         || $resultado->content =~ /Fatal error: Uncaught exception/) {
32
33         print "Vulnerável!";
34     }
35 }
36 else {
37     print "Não vulnerável!";
38 }
39
```

11.4.1 Veja a estrutura condicional para identificar uma página com vulnerabilidade de Injeção de SQL.

11.5 EXECUTANDO O SCRIPT

Esse é o resultado do script.

A screenshot of a terminal window with a dark background and light text. The window title is 'root@kali: /usr/bin'. The terminal shows the following commands and output:

```
File Edit View Search Terminal Help
root@kali:/usr/bin# perl tool.pl
Por favor, informe a url:
http://localhost/index.php?id=1

Vulnerável!
root@kali:/usr/bin#
```

12.0 IMPLEMENTAÇÕES

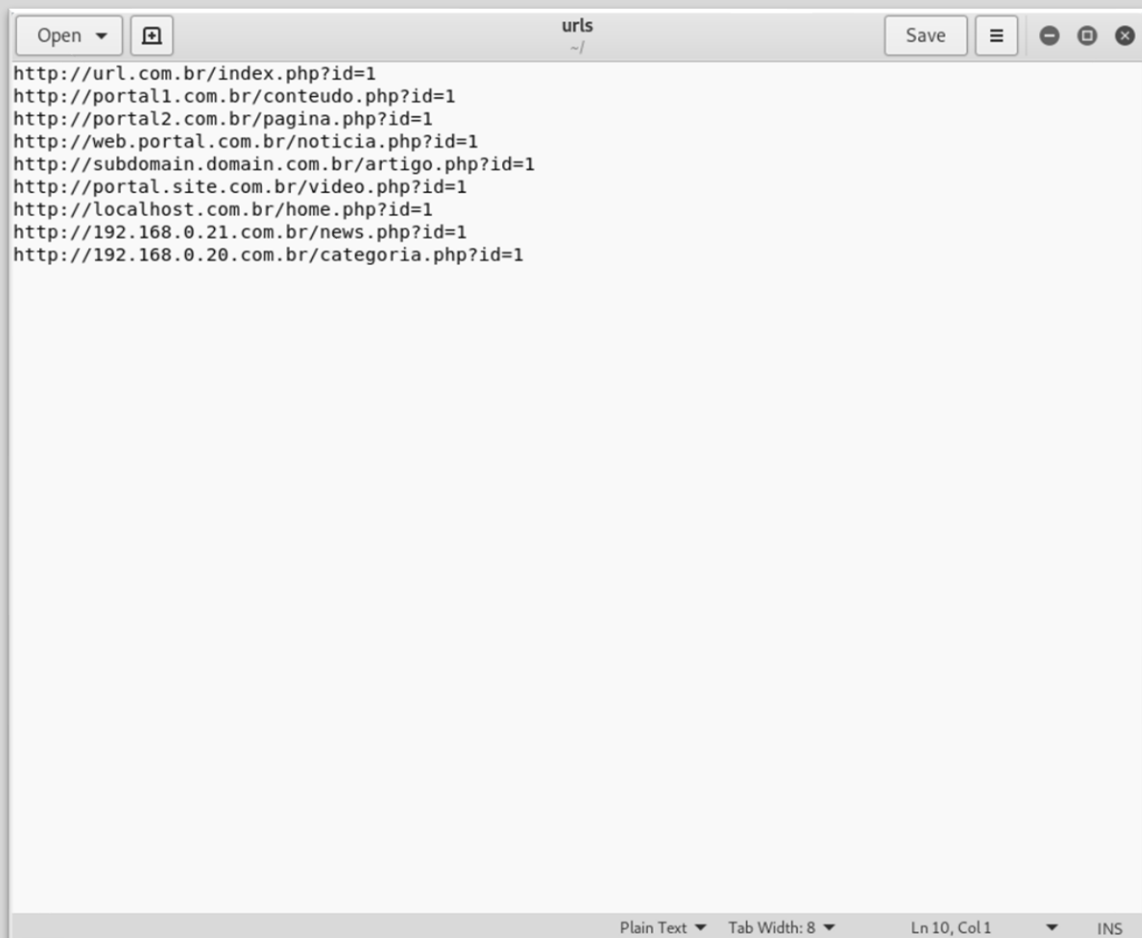
Algumas empresas possuem diversos sistemas, portais, sites internos e externos. Podemos adicionar todas as urls em um arquivo texto e alimentar o script **tool.pl**.

No momento que o script solicita o endereço da url, podemos substituir por uma função que solicite o nome do arquivo com as urls internas ou externas a serem testadas.

Esse processo facilita a execução de verificações e economiza tempo do analista de segurança.

Não precisa executar o script várias vezes, uma única execução é o suficiente para analisar vários endereços.

Abaixo, estou apresentando um modelo de arquivo texto com as urls a serem testadas como exemplo:



12.0.1 Lista de urls de sistemas, portais, sites internos e externos.

```
9
10  open( URL, "< $url" ) or die( "Nao foi possível abrir o arquivo: $!" );
11
12  our @vector = <URL>;
13
14  $last = $#vector;
15
16  for ( $i = 0; $i <= $last; $i++ ) {
17
18      $url = $vector[$i];
19
20      print "Verificando => ".$url."\n";
21
```

12.0.2 Na **linha 10** adicione o código responsável por ler o arquivo texto com urls.

A **linha 12** armazena todo o conteúdo do arquivo no array **@vector**.

Na **linha 14** acessamos o último elemento do array.

Na **linha 16** desenvolvemos um for para acessar cada linha ou url armazenado no arquivo.

Na **linha 18**, temos o armazenamento de cada endereço na variável **\$url**.

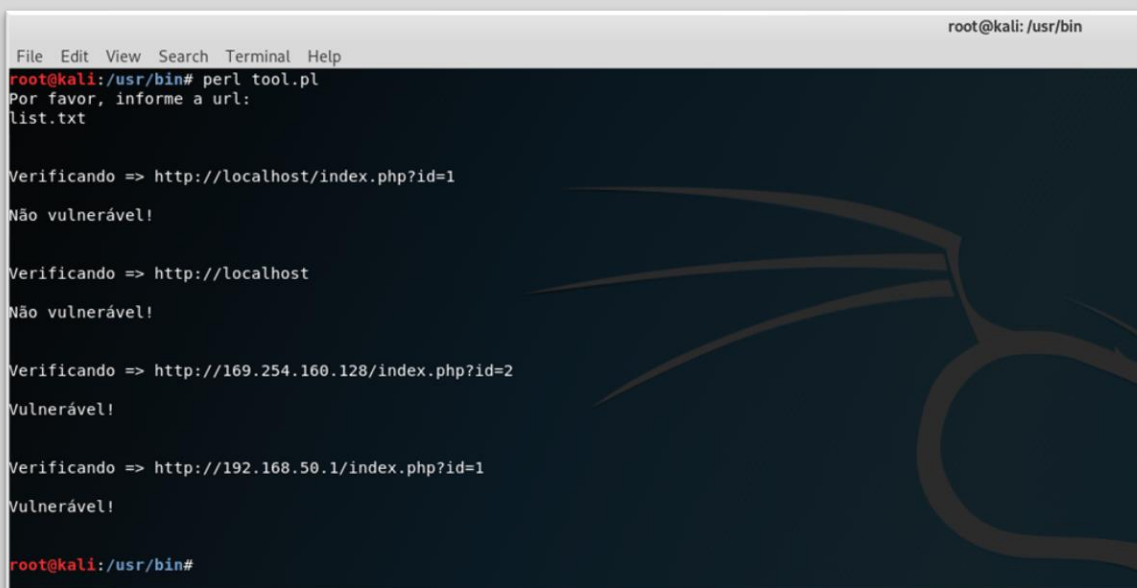
```

21
22     if ($resultado->content =~ /You have an error in your SQL syntax/
23     || $resultado->content =~ /Server Error in/
24     || $resultado->content =~ /SQL command not properly ended/
25     || $resultado->content =~ /PDOException: ERROR: unterminated quoted string at or near/
26     || $resultado->content =~ /Query failed: ERROR: syntax error at or near/
27     || $resultado->content =~ /Microsoft SQL Native Client error/
28     || $resultado->content =~ /[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value/
29     || $resultado->content =~ /Microsoft OLE DB Provider for ODBC Drivers/
30     || $resultado->content =~ /Microsoft JET Database Engine error/
31     || $resultado->content =~ /Fatal error: Uncaught exception/) {
32
33         print "Vulnerável!";
34
35     }
36     else {
37         print "Não vulnerável!";
38     }
39

```

12.0.3 Na linha 22 criamos um IF, que verifica se a resposta da página possui algum erro de banco de dados. Se houve algum erro, ele apresenta a imagem **“Vulnerável”** ou senão, **“Não vulnerável”**.

12.1 EXECUTANDO O SCRIPT



```

root@kali: /usr/bin
File Edit View Search Terminal Help
root@kali: /usr/bin# perl tool.pl
Por favor, informe a url:
list.txt

Verificando => http://localhost/index.php?id=1
Não vulnerável!

Verificando => http://localhost
Não vulnerável!

Verificando => http://169.254.160.128/index.php?id=2
Vulnerável!

Verificando => http://192.168.50.1/index.php?id=1
Vulnerável!

root@kali: /usr/bin#

```

12.1.2 Resultado do script verificando cada url armazenada no arquivo texto.

Você poderá adicionar novos recursos no script, como Thread, crawlers de páginas etc.

13.0 CÓDIGO COMPLETO

Abaixo é apresentado ambos os códigos em Perl para testar no seu ambiente particular.

13.1 CÓDIGO COMPLETO DAS FERRAMENTAS

Nessa seção temos a ferramenta para identificar vulnerabilidades, utilizando recursos simples de um scanning de vulnerabilidade.

```
#!/usr/bin/perl

use LWP::UserAgent;
use HTTP::Request;
use LWP::Simple;

print "Por favor, informe a url: \n";
$url = <stdin>;

$url = "http://localhost/index.php?id=1";

$url = $url."''";

my $requisicao=HTTP::Request->new(GET=>$url);

my $ua=LWP::UserAgent->new();
exit;

$ua->timeout(15);

my $resultado=$ua->request($requisicao);

if ($resultado->content =~ /You have an error in your SQL syntax/) {

    print "\n\n Vulnerável! \n\n";

}
else {
    print "\n\n Não vulnerável! \n\n";
}
```

13.2 A FERRAMENTA COM VÁRIAS VERIFICAÇÕES DE FALHAS EM BANCO DE DADOS

Nessa seção recebemos a resposta da requisição e verificamos com vários erros de banco de dados.

```
#!/usr/bin/perl

use LWP::UserAgent;
use HTTP::Request;
use LWP::Simple;

print "Por favor, informe a url: \n";
$url = <stdin>;

$url = "http://localhost/index.php?id=1";

$url = $url."''";

my $requisicao=HTTP::Request->new(GET=>$url);

my $ua=LWP::UserAgent->new();
exit;

$ua->timeout(15);

my $resultado=$ua->request($requisicao);

if ($resultado->content =~ /You have an error in your SQL syntax/
|| $resultado->content =~ /Server Error in/
|| $resultado->content =~ /SQL command not properly ended/
|| $resultado->content =~ /PSQLError: ERROR: unterminated quoted string at or near/
|| $resultado->content =~ /Query failed: ERROR: syntax error at or near/
|| $resultado->content =~ /Microsoft SQL Native Client error/
|| $resultado->content =~ /[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error
converting the nvarchar value/
|| $resultado->content =~ /Microsoft OLE DB Provider for ODBC Drivers/
|| $resultado->content =~ /Microsoft JET Database Engine error/
|| $resultado->content =~ /Fatal error: Uncaught exception/) {

    print "\n\n Vulnerável! \n\n";
}
else {
    print "\n\n Não vulnerável! \n\n";
}
```

13.3 A FERRAMENTA COM IMPLEMENTAÇÕES

Nessa seção utilizamos um recurso no scanning para verificar vários IPs ou urls com vulnerabilidade de SQL injection.

Para fazer essa verificação será preciso somente passar ou informar um arquivo de texto com vários ips ou urls.

```
#!/usr/bin/perl

use LWP::UserAgent;
use HTTP::Request;
use LWP::Simple;

print "Por favor, informe a url: \n";
$url = <stdin>;

open( URL, "< $url" ) or die ( "Não foi possível abrir o arquivo: $!");

@vector = <URL>;

$last = $#vector;

for ($i = 0; $i <= $last; $i++) {

    $url = $vector[$i]."";

    print "verificando => ".$vector[$i]."\n";

    my $requisicao=HTTP::Request->new(GET=>$url);

    my $ua=LWP::UserAgent->new();
    exit;
    $ua->timeout(15);

    my $resultado=$ua->request($requisicao);

    if ($resultado->content =~ /You have an error in your SQL syntax/
    || $resultado->content =~ /Server Error in/
    || $resultado->content =~ /SQL command not properly ended/
    || $resultado->content =~ /PSQLError: ERROR: unterminated quoted string at or near/
    || $resultado->content =~ /Query failed: ERROR: syntax error at or near/
    || $resultado->content =~ /Microsoft SQL Native Client error/
    || $resultado->content =~ /[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error
    converting the nvarchar value/
    || $resultado->content =~ /Microsoft OLE DB Provider for ODBC Drivers/
```

```
|| $resultado->content =~ /Microsoft JET Database Engine error/  
|| $resultado->content =~ /Fatal error: Uncaught exception/) {  
  
    print "\n Vulnerável! \n\n\n\n";  
  
}  
else {  
    print "\n Não vulnerável! \n\n\n\n";  
}  
  
}
```

14.0 CORRIGINDO VULNERABILIDADE

Irei demonstrar alguns tipos de proteção contra SQL Injection numa aplicação web que possui a tecnologia PHP.

1º Validar dados de entrada

Se o dado que passa pelo parâmetro GET é inteiro, então sempre podemos validar se a variável é **int**.

```
if (isset($_GET['id'])){  
    $id = (int)$_GET['id']; // → Validando se o valor é inteiro
```

Validar se o dado é numérico

```
4  
5     $id = (int)$_GET["id"];  
6  
7     if (is_numeric($id)) {  
8
```

Recomendo criar validação de dados de entrada manualmente. Abaixo, estou compartilhando um método muito eficaz e que contempla caracteres unicodes, hexadecimais, base64 e caracteres normais.

2º Prepare a Query

Crie sua consulta usando nomes de parâmetros precedidos por dois pontos como espaços reservados

3º Crie a declaração preparada

```
$statement = $dbh->prepare($consulta);
```

4º Vincular os parâmetros à instrução preparada

Vincule seus parâmetros à consulta.

```
$statement->bindParam(':id', $var);
```

5º Fazer as consultas

```
$statement->execute();
```

Etapa 6: buscar o resultado

```
$busca = $statement->fetchColumn();
```


15.0 PROTEÇÃO COM WAF MODSECURITY OU NAXSI

Nessa etapa serei bastante rápido.

Caso você tenha um WAF como o ModSecurity e deseje fornecer uma proteção adicional, recomendo adicionar a seguinte regra para prevenir seu sistema contra de ataques SQL Injection:

https://github.com/SEC642/modsec/blob/master/rules/base_rules/modsecurity_crs_41_sql_injection_attacks.conf

Outro Web Application Firewall conhecido é o Naxsi, sua estrutura de regra é menor e mais simples, mas também muito eficiente durante os testes.

A regra está disponível em:

https://github.com/nbs-system/naxsi/blob/master/naxsi_config/naxsi_core.rules

Ambos Web Application firewall trabalharão como um paliativo, cujo intuito é impedir que queries associados a SQL Injection possam surtir efeito no sistema web atacado.

Ressaltando a melhor recomendação é a correção das vulnerabilidades, mencionado na seção 14.0 desse artigo.

16.0 SOBRE O AUTOR

Paper criado por Fernando Mengali no dia 01 de março de 2025.

LinkedIn: <https://www.linkedin.com/in/fernando-mengali-273504142/>