Academia de Ciências Sociais e Tecnologias

INSTITUTO DE ESTUDOS AVANÇADOS EM CIÊNCIAS,
ENGENHARIA E TECNOLOGIAS

SEGURANÇA DE SOFTWARE

Mestrado em Engenharia de Segurança de Redes de Computadores

Guia de Laboratório III

Ataques SQL Injection

Julho 2019

Nuno Santos

nuno.m.santos@tecnico.ulisboa.pt

Introdução

Neste guia de laboratório focamo-nos em mais uma classe de ataques muito comum e que geralmente traz consequências muito graves para a confidencialidade e / ou integridade dos dados geridos por aplicações web: *SQL injections*. Temos por objectivos principais deste guia mostrar em que consistem estas vulnerabilidades, compreender os mecanismos subjacentes que estão na sua origem, e aprender como é que estas vulnerabilidades podem ser exploradas de tal forma que um atacante possa extrair ou mesmo modificar dados de uma aplicação web.

Para atingir estes objectivos, iremos realizar um conjunto de tarefas experimentais tendo por base o sistema operativo Linux e considerando aplicações web que utilizam bases de dados SQL para manter os seus dados persistentes. A primeira tarefa irá introduzir-nos às bases de dados SQL, em particular aos comandos necessários para aceder aos registos de uma base de dados. Este conhecimento é fundamental para compreender como é que um ataque SQL injection pode ser realizado. Na segunda tarefa iremos mostrar como efectivamente conduzir ataques SQL injection tendo por base uma aplicação web exemplo que tem vulnerabilidades. Esses ataques irão permitir extrair e modificar informação sobre os utilizadores dessa aplicação de forma indevida. Cada tarefa está dimensionada sensivelmente para ser realizada em duas horas e inclui um conjunto de exercícios práticos.

Contents

1 Tarefa 1: Introdução a Bases de Dados SQL

Nesta primeira tarefa iremos perceber de que forma é que as aplicações web interagem com bases de dados SQL e, mais importante, iremos apresentar um pequeno tutorial sobre os principais comandos SQL que as aplicações web actuais utilizam para interagir com essas bases de dados. Perceber a semântica destes comandos é fundamental para conseguirmos compreender posteriormente como surgem as vulnerabilidades SQL injection e de que forma é que um atacante pode explorar essas vulnerabilidades para extrair registos da base de dados ou para os modificar de forma não autorizada. No nosso percurso, começamos por apresentar uma base de dados de uma aplicação exemplo. Utilizando este exemplo, iremos depois mostrar como efectuar leituras e escritas numa base de dados SQL.

1.1 Base de Dados numa Aplicação Web Exemplo

Regra geral, uma aplicação web segue a arquitectura representada na Figura 1 e é constituida tipicamente por três componentes: um cliente que se localiza no browser e que serve de interface com o utilizador, o servidor web onde corre a lógica da aplicação que trata os pedidos enviados pelo browser, e a base de dados que é responsável por manter os dados da aplicação em armazenamento persistente. Para efeitos do nosso estudo, vamos focar-nos em bases de dados SQL, ou seja, bases de dados cuja interface permite o tratamento de comandos na linguagem SQL para gestão da base de dados.

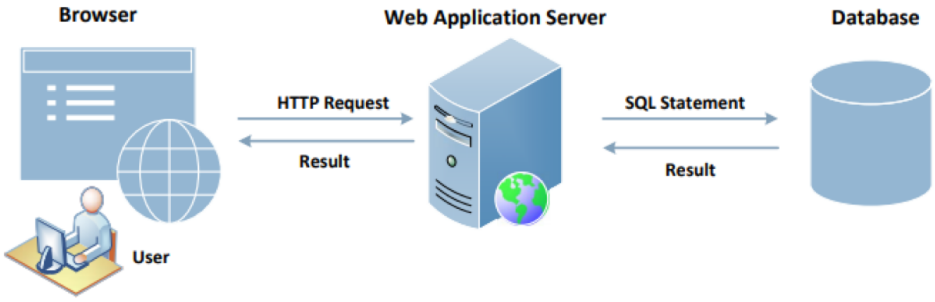

Figure 1: Arquitectura de uma aplicação web, em que uma base de dados SQL guarda estado persistente.

Para estudarmos como funciona a interação com uma base de dados SQL, vamos socorrer-nos de uma aplicação web exemplo que já está instalada na máquina virtual do laboratório: SEEDEmpregados. Na Tarefa 2 iremos aprender como interagir com esta aplicação através do cliente (isto é, pelo browser). Por agora, interessa-nos estudar como interagir com a base de dados desta aplicação directamente, ou seja, através da sua interface SQL.

Passo 1. Descrição e conteúdo da base de dados. Em primeiro lugar, dediquemos algumas palavras para perceber melhor em que consiste esta base de dados. Esta, tem por objectivo preservar informação sobre uma aplicação de gestão de empregados numa organização. Através dessa aplicação, os empregados podem consultar e modificar os seus dados pessoais. Existem dois papéis principais para os utilizadores dessa aplicação: `Administrator` é o papel priviligiado que pode gerir os perfis relativos à informação individual de cada empregado, e `Employee` é o papel normal atribuído ao empregado que pode consultar ou modificar o seu próprio perfil.

A informação dos empregados está descrita na tabela da Figura 2. Existem 6 registos (linhas) relativas a cada um dos utilizadores da aplicação. Um deles é o administrador, os restantes são empregados. Cada registo tem 10 campos (colunas) que representam diferentes tipos de informação sobre cada utilizador, como por exemplo, o nome, identificador, password, salário, etc.

Passo 2. Aceder à base de dados. Vamos agora aprender como se acede à base de dados através da linha de comandos. Isto será feito com a ajuda de uma ferramenta auxiliar chamanda `mysql`. Abra um terminal, e escreva o seguinte comando:

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

Figure 2: Tabela que mostra conteúdo da base de dados da aplicação SEEDEmpregados.

```
$ mysql -u root -pseedubuntu
```

Este comando permite a um utilizador pre-registado *root* aceder à base de dados. Esse utilizador tem que introduzir uma password (`seedubuntu`). Depois de estarmos logados ao servidor que gere as bases de dados do sistema, podemos começar a introduzir alguns comandos de gestão. Um destes permite listar informações sobre as bases de dados que já estão criadas:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Users              |
| elgg_csrf          |
| elgg_xss           |
| mysql              |
| performance_schema |
| phpmyadmin         |
| sys                |
+--------------------+
8 rows in set (0.01 sec)
```

Interessa-nos a base de dados `Users` pois é aquela que está associada à aplicação web SEEDEmpregados. Podemos então selecionar essa base de dados da seguinte forma:

```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Depois disto, podemos listar quais são as tabelas que foram criadas na base de dados `Users`:

```
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)
```

Vemos que existe uma tabela chamada `credential`. Agora, se quisermos consultar o conteúdo desta tabela, e em especial verificar se apresenta os dados listados na Figura 2, temos que utilizar um comando SQL específico e que veremos já de seguida.

1.2 Operações de Leitura da Base de Dados

Para lermos registos de uma base de dados, é necessário utilizar um comando SQL chamado SELECT.

Passo 1. Listar todas as colunas. O exemplo em baixo mostra como listar todo o conteúdo da base de dados `credential` (isto é, a base de dados da nossa aplicação exemplo):

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
    | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
         |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
         |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |
         |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |
         |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |
         |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |
         |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
6 rows in set (0.01 sec)
```

Apesar do resultado parecer confuso dado que a largura da tabela é maior do que a largura da linha de texto, vemos que o conteúdo da tabela na base de dados reflecte a tabela da Figura 2.

Passo 2. Seleccionar as colunas. O asterisco no comando SELECT dá indicação para ler todos os campos (colunas) da base de dados. Se quisermos seleccionar apenas algumas colunas específicas, podemos fazê-lo substituindo o asterisco pelo nome de uma dessas colunas, ou por uma lista de nomes de colunas separadas por vírgulas. Por exemplo, para listar as colunas com o nome (`Name`) e salário (`Salary`), podemos correr um comando SELECT da seguinte forma:

```
mysql> select Name, Salary from credential;
+-------+--------+
| Name  | Salary |
+-------+--------+
| Alice |  20000 |
| Boby  |  30000 |
| Ryan  |  50000 |
| Samy  |  90000 |
| Ted   | 110000 |
| Admin | 400000 |
+-------+--------+
6 rows in set (0.00 sec)
```

Passo 3. Seleccionar as linhas. Além de listar todas as linhas, ou seja todos os registos da base de dados, é possível restringir os registos que devem ser mostrados em função de um determinado critério. Isto é feito utilizando uma cláusula chamada WHERE. Por exemplo, considere o comando seguinte:

```
mysql> select * from credential where Name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
    | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
         |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
1 row in set (0.00 sec)
```

Este comando mostra todas as colunas dos registos cuja coluna `Nome` seja igual a `Alice`. Como só existe um utilizador com esse nome, só um registo é mostrado. Com base no que já aprendemos acima, podemos restringir também as colunas, mostrando só os campos nome (`Name`) e salário (`Salary`):

```
mysql> select Name, Salary from credential where Name='Alice';
+-------+--------+
| Name  | Salary |
+-------+--------+
| Alice |  20000 |
+-------+--------+
1 row in set (0.00 sec)
```

Passo 5. Operações lógicas para selecção de linhas. O SQL permite critérios mais ricos para selecção de linhas, em especial com a utilização de operadores lógicos. Por exemplo, listar todos os registos cujo salário é inferior a 70000.

```
mysql> select Name, Salary from credential where Salary<70000;
+-------+--------+
| Name  | Salary |
+-------+--------+
| Alice |  20000 |
| Boby  |  30000 |
| Ryan  |  50000 |
+-------+--------+
3 rows in set (0.00 sec)
```

Ou então, compondo multiplas proposições lógicas, por exemplo, mostrando todos os registos cujo salário é inferior a 70000 mas superior a 40000.

```
mysql> select Name, Salary from credential where Salary<70000 and Salary>40000;
+------+--------+
| Name | Salary |
+------+--------+
| Ryan |  50000 |
+------+--------+
1 row in set (0.00 sec)
```

Também podemos utilizar operações lógicas dijuntivas para combinar registos que obedeçam a vários critérios. Por exemplo, interprete o que faz o comando seguinte:

```
mysql> select Name, EID, Salary from credential where EID=10000 or Name='Boby';
+-------+-------+--------+
| Name  | EID   | Salary |
+-------+-------+--------+
| Alice | 10000 |  20000 |
| Boby  | 20000 |  30000 |
+-------+-------+--------+
2 rows in set (0.00 sec)
```

Passo 6. Uma curiosidade. No comando acima, o que acontece ao adicionar a condição "`or 1=1`"?

```
mysql> select Name, EID, Salary from credential where EID=10000 or Name='Boby'
    or 1=1;
+-------+-------+--------+
| Name  | EID   | Salary |
+-------+-------+--------+
| Alice | 10000 |  20000 |
| Boby  | 20000 |  30000 |
| Ryan  | 30000 |  50000 |
| Samy  | 40000 |  90000 |
| Ted   | 50000 | 110000 |
| Admin | 99999 | 400000 |
```

```
+-------+-------+--------+
6 rows in set (0.00 sec)
```

Aparecem todos os registos! Como "1=1" é sempre verdadeiro, então aplicando a operação de disjunção à condição anterior, o resultado passa a ficar sempre verdadeiro. Quer dizer que a condição "`or` 1=1" tem por efeito anular todas as outras condições anteriores por forma a mostrar todos os registos. Consegue pensar numa condição oposta que tenha por finalidade cancelar a listagem de todos os registos?

Passo 7. Comentários. Antes de passarmos às operações de escrita vale a pena mostrar como se especificam comentários em comandos SQL. Veremos sobretudo na Tarefa 2 como isto é importante para lançar ataques SQL injection. Existem três formas de introduzir comentários. As linhas seguintes mostram estas três formas e resultam todas na mesma operação:

```
mysql> select * from credential; # Comentario no fim da linha
mysql> select * from credential; -- Comentario no fim da linha
mysql> select * /* Comentario no meio da linha */ from credential;
```

1.3 Operações de Escrita na Base de Dados

Agora que vimos como podemos ler de uma base de dados, vejamos como efectuar escritas. Existem três tipos de operações de escritas: modificar um registo (UPDATE), inserir um registo (INSERT), ou apagar um registo (DELETE). Vejamos cada uma de cada vez.

Passo 1. Modificar um registo. A modificação de um registo é feita com o comando UPDATE. Por exemplo, o salário do Boby é 30000. Actualizemo-lo para 82000:

```
mysql> update credential set Salary=82000 where Name='Boby';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Para verificar que o comando foi bem sucedido, volte a executar um comando SELECT para listar o registo do Boby. No comando UPDATE, imediatamente à frente da palavra `set` indicamos qual é o campo que queremos modificar. Depois é possível ter ainda a cláusula WHERE que nos permite restringir as linhas que queremos modificar.

Passo 2. Inserir um registo. Vejamos agora como inserir um novo registo na base de dados com o comando INSERT. Tente perceber o que faz o comando seguinte:

```
mysql> insert into credential (Name, EID, Salary) values ('Nuno', '60000',
    '10000');
Query OK, 1 row affected (0.01 sec)
```

Passo 3. Apagar um registo. Vejamos a última operação de escrita de registos que permite apagar um registo. É feito utilizando o comando DELETE. Por exemplo, corra o comando seguinte para apagar o registo que foi inserido pela operação anterior:

```
mysql> delete from credential where Name='Nuno';
Query OK, 1 row affected (0.01 sec)
```

Verifique que este registo foi efectivamente apagado da base dados.

1.4 Exercícios

Façamos agora alguns exercícios com o fim de ganharmos alguma experiência na interação com bases de dados SQL e consolidar os conceitos apresentados até agora.

Exercício 1. Utilizado a mesma base de dados da aplicação exemplo, efectue uma pesquisa que mostre o EID de todos os utilizadores com um salário superior a 70000 e que não sejam o Admin. Nota: o operador "diferente" é indicado por "<>".

Exercício 2. Descubra o que faz o comando seguinte e depois execute-o para verificar se o resultado estava de acordo com a sua descrição.

```
mysql> select Nome, EID from credential where Name='Alice' or Name='Ted' or 1=1;
```

Exercício 3. Considerando o comando SQL indicado em baixo, diga se concorda com a afirmação seguinte, explicando as suas razões: "Este comando actualiza o campo de salário dos utilizadores Alice e Ted colocando os seus valores a 1."

```
mysql> update credential set Salary='1' where /* Name='Alice' or */ Name='Ted';
```

Exercício 4. Estude o comando seguinte e descreva o que faz. Depois teste-o para verificar a sua hipótese:

```
mysql> select * from credential where Name='Admin'; # and Password='123';
```

2 Tarefa 2: Ataque SQL Injection

Na primeira tarefa vimos de que forma as bases de dados são utilizadas para armazenamento persistente de informação no contexto de aplicações web. Vimos também que as bases de dados são acedidas através de uma linguagem especial chamada SQL e aprendemos alguns comandos fundamentais SQL que permitem ler registos das bases de dados e modificar ou apagar esses registos. É essencial para nós compreender como funcionam estes comandos pois estão na base de uma classe de vulnerabilidades que permite um conjunto de ataques muito comuns (e sérios) chamados *injecção SQL*. Nesta tarefa iremos aprender como podem ser lançados esses ataques para extracção ou modificação de registos de uma base de dados. Mas antes disso precisamos aprender em mais pormenor como se processa a interação de uma aplicação web com a própria base dados. Iremos utilizar a mesma aplicação web que vimos na Tarefa 1 e que permite gerir empregados numa organização: a aplicação SEEDEmpregados. Pode ser acedida através do browser no endereço: `www.seedlabsqlinjection.com`.

2.1 Interacção da Aplicação Web com a Base de Dados

Em geral, a arquitectura de uma aplicação web está representada na Figura 3. O utilizador utiliza o browser como cliente para interagir com a aplicação. Este envia pedidos HTTP para um servidor web, que processa esses pedidos de acordo com a lógica da aplicação, e se necessário envia por sua vez pedidos SQL à base de dados por forma a servir esses pedidos. Para percebermos melhor este processo, vamos estudar em detalhe como é feito o processo de *login* quando o utilizador se autentica perante a aplicação SEEDEmpregados. Estudemos este processo em vários passos.

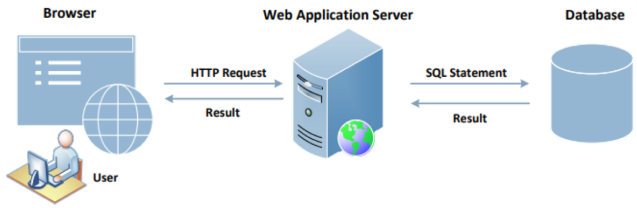

Figure 3: Arquitectura de uma aplicação web.

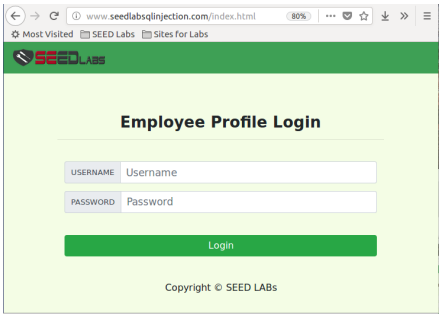

Figure 4: Ecrã login.

Passo 1. Obter dados do utilizador. O primeiro passo consiste em aceder à página principal da aplicação SEEDEmpregados por forma a solicitar o nome de utilizador e respectiva password ao utilizador. Comece por aceder ao site `www.seedlabsqlinjection.com` no browser e introduza as credentiais de acesso do utilizador Alice (nome: `Alice`, password: `seedalice`, mas ainda não carregue no botão "`Login`". O aspecto da página pode ser observado na Figura 4.

Para entender o que acontece quando carregamos no botão "`Login`", altura em que os dados são enviados para o servidor web, podemos ver o código fonte da página HTML que o browser nos está a mostrar. vá ao menu "*Tools -> Web Developer -> Page Source*" e verifique se encontra o seguinte fragmento código HTML, abreviado:

```
<form action="unsafe_home.php" method="get">
      ...
      <input type="text" class="form-control" placeholder="Username" name="
          username" aria-label="Username" aria-describedby="uname">
      ...
      <input type="password" class="form-control" placeholder="Password" name="
          Password" aria-label="Username" aria-describedby="pwd">
      ...
    <button type="submit" class="button btn-success btn-lg btn-block">Login</
        button>
```

```
  </form>
```

Estas linhas são responsáveis por criar um formulário que vai ler dois valores introduzidos pelo utilizador: um de tipo *text* e que representa o nome do utilizador `username`, e o outro do tipo *password* e que denota a palavra passe introduzida pelo utilizador (este parâmetro tem o nome `Password`).

Passo 2. Envio do pedido HTTP. Para vermos o pedido HTTP que é submetido ao servidor quando pressionamos o botão "`Login`", abra primeiro a janela lateral "HTTP Live Headers". Depois carregue no botão e anote qual foi o resultado obtido. Confirme que o browser enviou o pedido para o URL seguinte:

```
http://www.seedlabsqlinjection.com/unsafe_home.php?username=Alice&Password=
    seedalice
```

Repare como neste pedido, há uma correspondência directa entre os nomes dos parâmetros do formulário e os respectivos valores introduzidos pelo utilizador, com os parâmetros que são colocados no final do URL, depois do caracter "?". O servidor web vai processar este pedido, e se correr tudo bem, vai ver o ecrã mostrado na Figura 5.

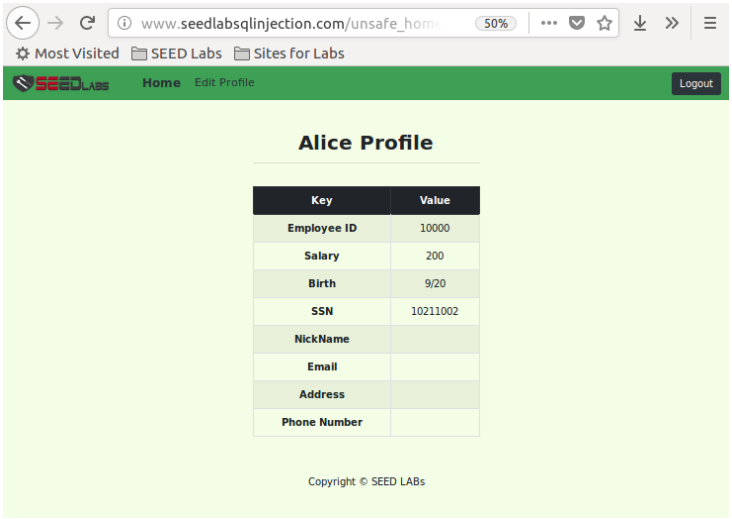

Figure 5: Aplicação SEEDEmpregados depois do login bem sucedido da Alice.

Passo 3. Como o servidor web processou o pedido. Quando o browser enviar o pedido HTTP para o URL indicado acima, podemos ver, olhando para o URL, que o pedido é direccionado à página `unsafe_home.php`. Esta página, na realidade, é um programa escrito na linguagem PHP que vai processar o pedido, e que consiste em verificar se o utilizador e respectiva password são válidos, e depois vai gerar dinamicamente uma nova página HTML que vai ser enviada para o browser. É esta página que está representada na Figura 5 e que mostra ao utilizador o resultado da operação.

Vejamos então o que faz o ficheiro `unsafe_home.php`. Abra esse ficheiro, que está localizado na pasta `/var/www/SQLInjection`, e procure as linhas seguintes:

```
<?php
session_start();
// if the session is new extract the username password from the GET request
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
```

Nestas linhas vemos como o código PHP acede aos campos `username` e `Password` que são enviadas como parâmetros do pedido HTTP. Estes campos são acedidos e guardados, respectivamente em duas variáveis: `$input_uname` e `$input_pwd`. A variável `$hashed_pwd` guarda um sumário seguro da pass-

word. Algumas linhas mais abaixo nesse ficheiro, vemos que estes campos são usados por um comando SQL que é enviado à base de dados por forma a confirmar se estes dois campos estão correctos ou não:

```
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
               email,nickname,Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
...
  if($id!=""){
  // If id exists that means user exists and is successfully authenticated
  drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address
      ,$phoneNumber);
}else{
  // User authentication failed
  ...
  return;
}
```

Repare que no comando SQL (que fica guardado na variável `$sql`), as variáveis `$input_uname` e `$hashed_pwd` vão ser substituídas pelos campos que foram passados pelos utilizador no formulário, nomeadamente o seu username e sumário (*hash*) da password. Para verificar o que é que a base de dados vai retornar, efectue este comando directamente na consola `mysql`. Use como `$hashed_pwd` da Alice, o valor: `fdbe918bdae83000aa54747fc95fe0470fff4976`. Para este fim, siga as instruções descritas na Tarefa 1.

2.2 Ataque SQL Injection para Extracção de Informação

Como vimos no código PHP apresentado na secção anterior, o comando SQL enviado para o servidor é construído com base no nome do utilizador e na password introduzidos pelo utilizador aquando da operação de login na aplicação web SEEDEmpregados. Mas o que acontece se o utilizador, em vez de escrever estes valores, enviar um conjunto de caracteres tais que, quando substituídos no comando SQL, estes alteram a semântica do comando SQL original? Diz-se que foi feito um ataque de injecção de código SQL, ou *SQL injection*. Agora, vamos demonstrar como é possível lançar um ataque deste tipo de forma a extrair informação da base dados à qual não temos acesso directo a partir da interface web.

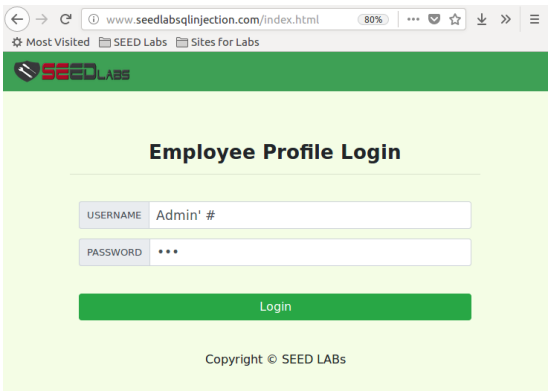

Figure 6: SQL injection para aceder como `Admin`.

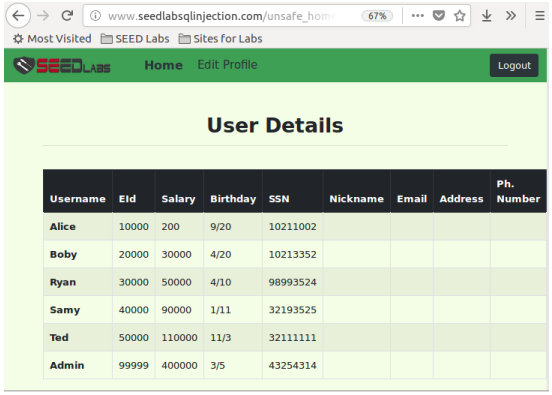

Figure 7: Portal privado do administrador.

Passo 1. Lançar o ataque. Comecemos de imediato executando um ataque SQL injection. O nosso objectivo é conseguirmos logar-nos como administradores do sistema e aceder a toda a informação sobre os outros empregados, mas com a seguinte dificuldade: sabemos que o username do administrador é `Admin`, mas não sabemos qual é a sua password. Para lançar o ataque, visite a página de login da aplicação web, e preencha-a tal como indicado na Figura 6, ou seja: username="`Admin' #`" e password="`123`".

(A password pode ser qualquer coisa.) Pressione o botão "Login" e verifique que teve acesso ao portal do administrador (ver Figura 7)! Eis os detalhes de todos os empregados! Confirme que esta é a vista do administrador voltando a logar-se, mas agora com a sua verdadeira password: Analise o texto que foi introduzido: explique porque razão conseguimos aceder como administrador sem saber a sua password?

Passo 2. Explicação do ataque. Para entendermos o que se passou, ajuda fazer o exercício de substituir, no comando SQL do ficheiro PHP, os valores do username e da password, pelos valores que acabamos de construir no ataque. Se fizermos isso, o resultado é algo do género:

```
SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
    email,nickname,Password
FROM credential
WHERE name= 'Admin' #' and Password='<hash da password>'";
```

Acontece que o texto que enviámos nos campos do formulário, em especial no campo username, vão alterar o comando SQL original. No campo do username introduzimos "Admin' #" com o seguinte objectivo. Dado que o caracter "#" permite introduzir comentários em SQL, ao colocarmos este caracter estamos a comentar tudo o que está à imediatamente frente no comando SQL, o que significa que já não vamos pedir à base de dados para validar a password. O caracter "'" simplesmente fecha o primeiro "'" que já estava no comando SQL original, e Admin é o nome do utilizador com que queremos aceder.

Passo 3. Logar-se como outro utilizador. Para consolidar o que aprendemos, repita agora o mesmo ataque, mas aceda à conta de outro utilizador – o Boby.

2.3 Ataque SQL Injection para Modificação de Informação

O ataque SQL injection que vimos anteriormente tirava partido de uma vulnerabilidade num comando SQL que permite ler da base de dados – é um comando SELECT. Vamos ver agora que é possível seguir o mesmo método, mas agora para introduzir modificações na base de dados. Isto acontece se conseguirmos fazer um ataque de SQL injection em comandos SQL que usem a primitiva UPDATE ou INSERT INTO. Por sorte, a nossa aplicação web SEEDEmpregados tem uma página que permite ao utilizador modificar o ser perfil pessoal. Será que conseguimos lançar um ataque para modificar o seu salário?

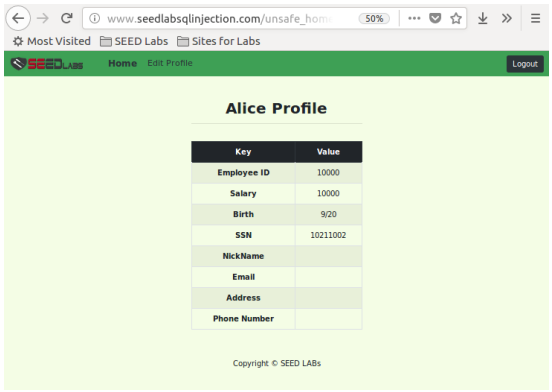

Figure 8: Página do perfil da Alice.

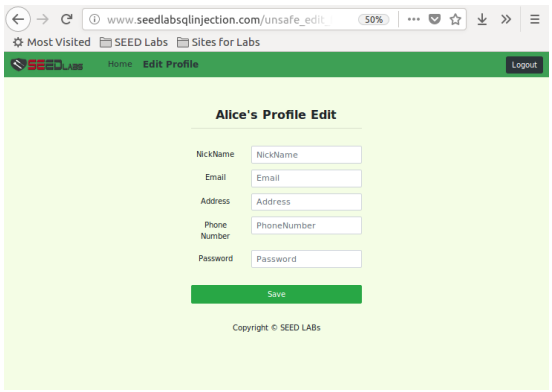

Figure 9: Página de edição de perfil da Alice.

Passo 1. Investigar a página de edição de perfil. Comecemos por investigar a página que permite a um utilizador legítimo aceder ao seu perfil pessoal. Autentique-se como o utilizador Alice (password: seedalice). Ao entrar na página principal pode ver o perfil da Alice (ver Figura 8 em que alguns dos seus dados são mostrados. Para modificar alguma desta informação, aceda à página de edição, clicando na opção "*Edit Profile*"' no topo dessa página. Deve observar uma nova página com o aspecto da Figura 9. Ali, pode modificar alguns campos – o *nickname*, o email, o endereço, o número de telefone,

e a password – mas , obviamente, não o salário. Se quiser, pode experimentar a mudar o número de telefone para verificar que essa informação foi actualizada no perfil.

Passo 2. Analisar o comando SQL para edição de perfil. Por forma a tentar identificar uma vulnerabilidade SQL injection que nos permita introduzir modificações na base de dados a partir da página de edição de perfil, será útil inspeccionarmos qual o ficheiro PHP que serve esta página. Para obter essa informação, uma forma é observar o código HTML fonte da página de edição e inspeccionar o atributo `action` associado ao formulário. Neste caso, esse atributo aponta para o ficheiro `unsafe_edit_backend.php` Com a ajuda de um editor, inspeccione esse ficheiro (está localizado na pasta `/var/www/SQLInjection`, local onde está instalada toda a aplicação web). Navegue nesse ficheiro à procura do comando SQL responsável pelo tratamento do pedido de actualização do perfil. Trata-se de um comando SQL UPDATE. Verifique que encontre as linhas seguintes:

```
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email
      ',address='$input_address',Password='$hashed_pwd',PhoneNumber='
      $input_phonenumber' where ID=$id;";
}else{
  // if passowrd field is empty.
  $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email
      ',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id
      ;";
}
$conn->query($sql);
```

Passo 3. Modificar o próprio salário. Já vimos que não é possível ao empregado modificar o seu próprio salário através da interface disponibilizada pela aplicação web. Vamos agora ver se conseguimos efectuar um SQL injection de forma a alterar o salário da Alice de 10000 para 20000 (um generoso aumento de 100%). Assuma que sabe que os salários estão guardados numa coluna com o nome `salary`. Façamos por partes:

- Para efectuar este ataque, começamos por ver qual o comando SQL UPDATE que iremos explorar (isto porque existem dois no código fonte acima). Se não modificarmos a password, o programa executa o segundo UPDATE (isto é, do *else*). Então, se quisermos abusar esse comando, não podemos mudar a password.
- Temos agora que pensar sobre o texto que temos que injectar num dos campos para ao mesmo tempo (a) introduzir o novo valor no salário que se pretende, e (b) sem estragar o resto do comando SQL. Pense um pouco na solução.

Uma hipótese possível seria, no campo *nickname*, introduzir o texto seguinte: "`a', salary='20000`" e pressionar no botão "`Save`". Experimente como faria se utilizasse outros campos em vez do *nickname*.

2.4 Exercícios

Agora que já sabemos como efectuar ataque SQL injection, vamos fazer mais um ataque na linha deste último ataque que teve por objectivo alterar o salário do próprio empregado.

Exercício 1. A Alice não gosta do Bob. O que ela pretende fazer é reduzir o salário do Bob para o valor 1. Execute um ataque SQL injection que obtenha esse resultado. Assuma que sabe que os nomes dos utilizadores estão guardados a coluna `name`.

3 Para Aprender Mais

Como tarefas opcionais, caso tenha conseguido terminar as tarefas anteriores em tempo útil, sugerimos que estude os mecanismos de mitigação de vulnerabilidades SQL injection. Remetemos o leitor interessado para o guia de laboratório da SEED Labs[1] para obter mais informações sobre estes mecanismos.

[1] https://seedsecuritylabs.org/Labs_16.04/PDF/Web_SQL_Injection.pdf