

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FEELT – FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA DE COMPUTAÇÃO

LUCAS ALBINO MARTINS
12011ECP022

**SEGURANÇA DE SISTEMAS COMPUTACIONAIS: SEEDLABS 20.04: WEB
SECURITY – SQL INJECTION ATTACK LAB**

UBERLÂNDIA
2021

Introdução.

O SQL-Injection é o nome dado a uma falha na codificação de uma aplicação qualquer (seja web ou local) que possibilita, por meio de um input qualquer, a manipulação de uma consulta SQL. Essa manipulação é chamada Injeção, então, o termo Injeção SQL. Resumindo: o SQL-Injection é uma técnica de ataque baseada na manipulação do código SQL, que é a linguagem utilizada para troca de informações entre aplicativos e bancos de dados relacionais.

As consequências de um SQL-Injection variam de leves a graves. Após uma injeção, um hacker pode:

- Corromper, roubar ou excluir dados.
- Obter acesso root ao sistema.
- Transformar o servidor em uma botnet.
- Elevar os privilégios para alcançar outros aplicativos e sistemas na rede.
- Comprometer o servidor ou outra infraestrutura de back-end.
- Lançar um ataque DDoS (negação de serviço).
- Acessar o sistema operacional por meio do servidor de banco de dados.
- A extensão do dano depende da habilidade do atacante. A vítima pode experimentar qualquer coisa, desde alguns erros de banco de dados até o controle total do servidor da web.

Na grande maioria dos aplicativos da web exigem que os usuários forneçam credenciais autenticadas para provar sua identidade e nível de acesso. Quando um usuário verificado solicitar dados, o aplicativo enviará uma instrução SQL ao banco de dados na forma de uma consulta e retornará os dados solicitados.

Então um aplicativo tem uma vulnerabilidade de SQL-Injection, um hacker pode pular o processo de autenticação e injetar manualmente instruções SQL (ou carga maliciosa) no banco de dados. O banco de dados não reconhece a ameaça e executa a instrução como se a própria aplicação estivesse enviando a solicitação.

De modo contrário de alguns tipos de ataques cibernéticos, o SQL-Injection requer que o sistema de destino tenha uma falha explorável. A maioria dos pontos fracos surge da falta de separação estrita entre o código do programa e a entrada fornecida pelo usuário.

Do ambiente do laboratório.

Seguindo todos os procedimentos a partir de um manual disponibilizado pelo site da SEED SECURITY, para aplicar os teste do laboratório sobre SQL-Injection foi necessário utilizar uma maquina virtual com ambiente Linux no caso a VirtualBox16.08(Oracle) e uma imagem disponibilizada pela equipe da SEED SECURITY SEED-Ubuntu20.04, todos links estão disponíveis em:

https://seedsecuritylabs.org/Labs_20.04/Web/Web_SQL_Injection/

A partir da VM já instalada então foi necessário inserir uma linha no arquivo /etc/hosts para que a pagina de login fosse acessada “10.9.0.5 www.seed-server.com”.

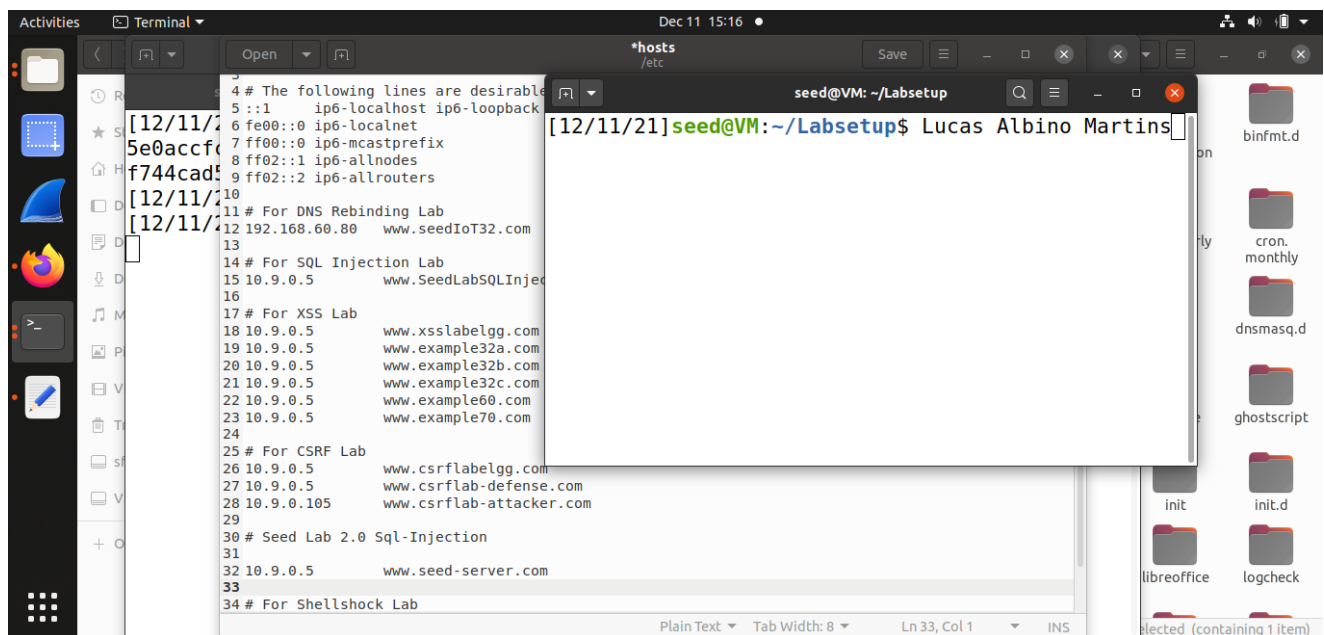
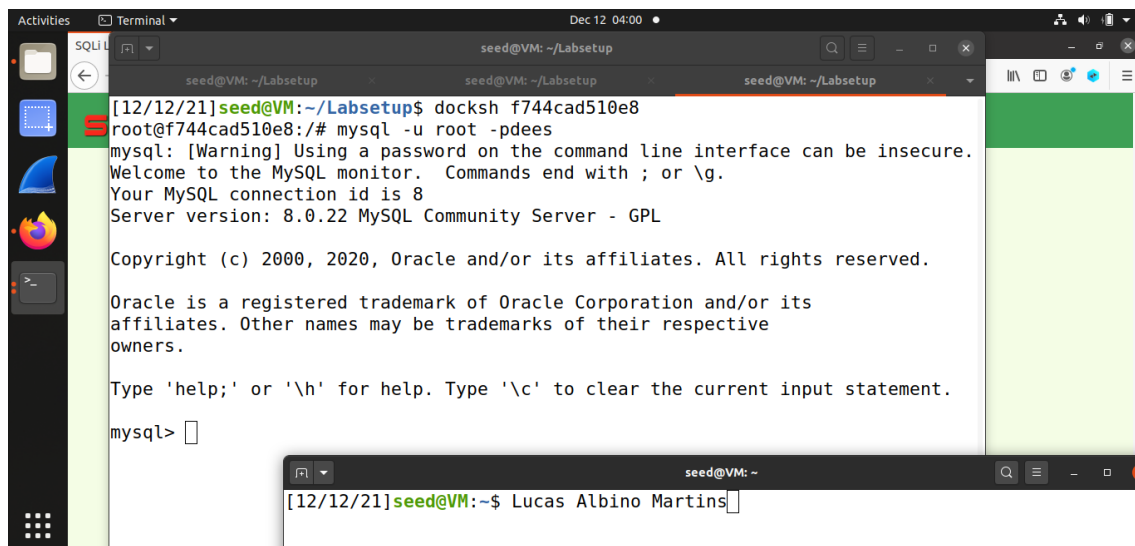


Imagem 1 – Gedit Hosts.

Foi configurado uma pasta de compartilhamento a partir do manual, então enviado para VM um arquivo para execução do laboratório seguindo o manual Labsetup.zip. Escolhido um local dentro da VM foi descompactado os arquivos e executado os comandos do docker “*docker-compose build e docker-compose up*” para subir a imagem do Docker-compose para criar o ambiente de testes web.



A terminal window titled 'seed@VM: ~/Labsetup' showing the execution of 'docksh f744cad510e8' to start a MySQL container. The output shows the MySQL command-line interface with a warning about using a password on the command line, a welcome message, and the MySQL version (8.0.22). The user 'root' is connected with password 'pdees'. The prompt 'mysql>' is visible.

```
[12/12/21]seed@VM:~/Labsetup$ docksh f744cad510e8
root@f744cad510e8:/# mysql -u root -pdees
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

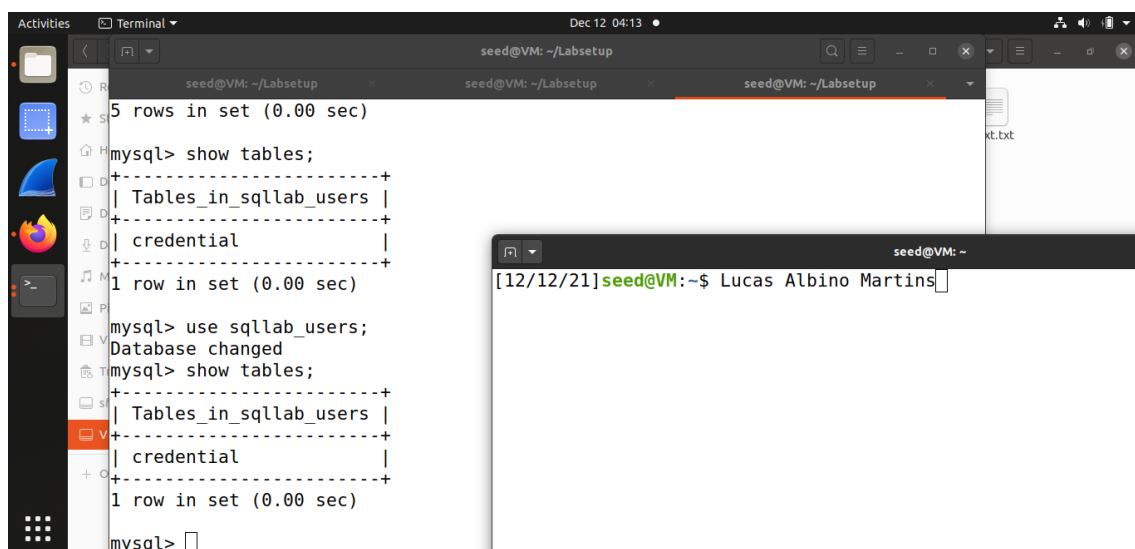
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Imagem 4 – Banco MYSQL.



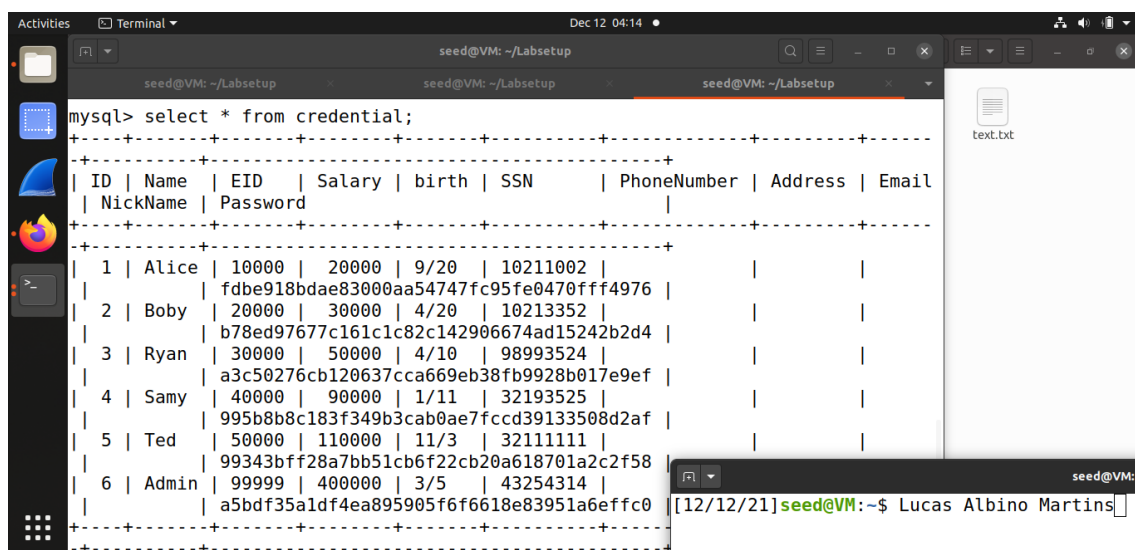
A terminal window titled 'seed@VM: ~/Labsetup' showing the execution of 'show tables;' and 'use sqllab_users;'. The output shows the tables in the 'sqllab_users' database: 'Tables_in_sqllab_users' and 'credential'. The prompt 'mysql>' is visible.

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

mysql> use sqllab_users;
Database changed
mysql> show tables;
+-----+
| Tables_in_sqllab_users |
+-----+
| credential              |
+-----+
1 row in set (0.00 sec)

mysql>
```

Imagem 5 – Tabela sqllab_users.



A terminal window titled 'seed@VM: ~/Labsetup' showing the execution of 'select * from credential;'. The output displays a table with columns: ID, Name, EID, Salary, birth, SSN, PhoneNumber, Address, Email, NickName, Password. The data is as follows:

| 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 |

Imagem 6 – Tabela de credenciais.

Task 2.1: SQL Injection utilizando um SELECT:

Na tela de login existem 2 campos, um campo de usuário (username) e outro de senha (password). Analizando a lógica dos campos podemos ver que é feito um WHERE diretamente no campo de username e password (que é convertido em sha1 e comparado com as senhas guardadas no banco).

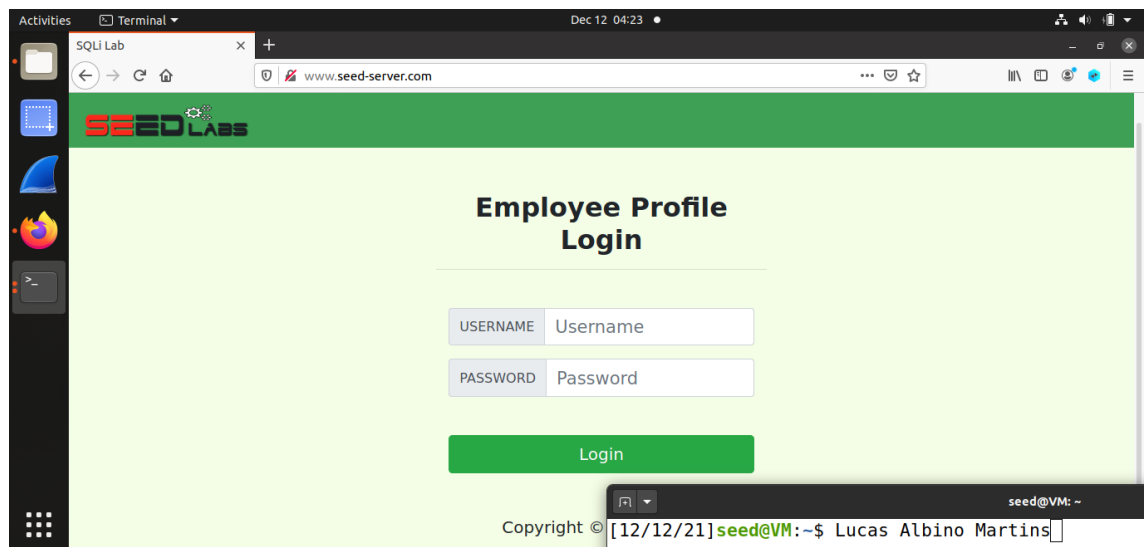


Imagem 7 – Tela de login.

```
$conn = getDB();  
// Don't do this, this is not safe against SQL injection attack  
$sql="";  
if($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 password field is empty.  
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$  
    input_email',address='$input_address',PhoneNumber='$input_phonenumber'  
    where ID=$id;";  
}
```

Imagem 8 – Campos de username e password.

Logo podemos por exemplo usar o seguinte login “admin’#”:

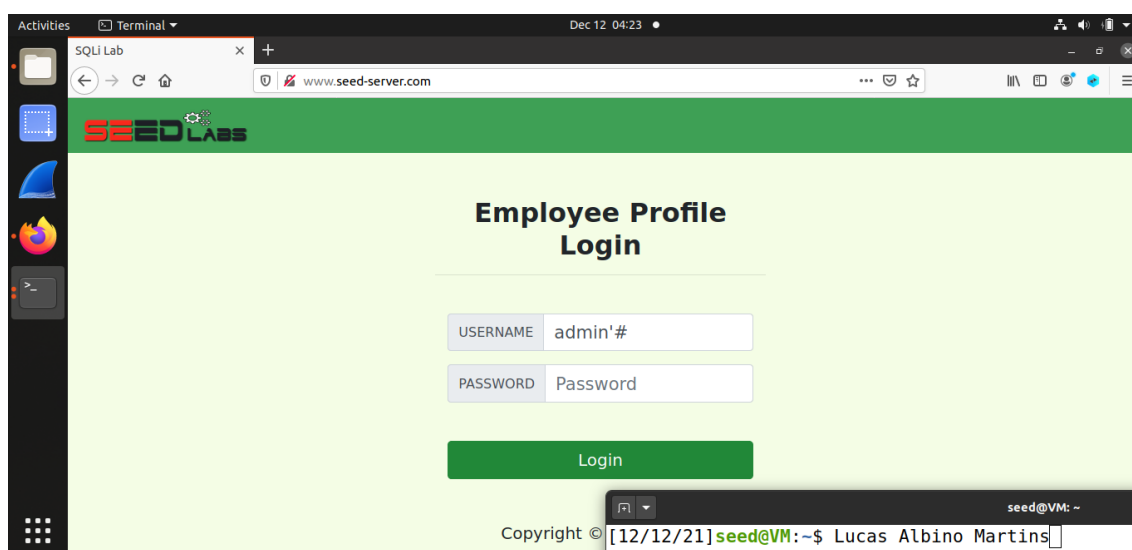


Imagem 9 – Utilizando login admin’#.

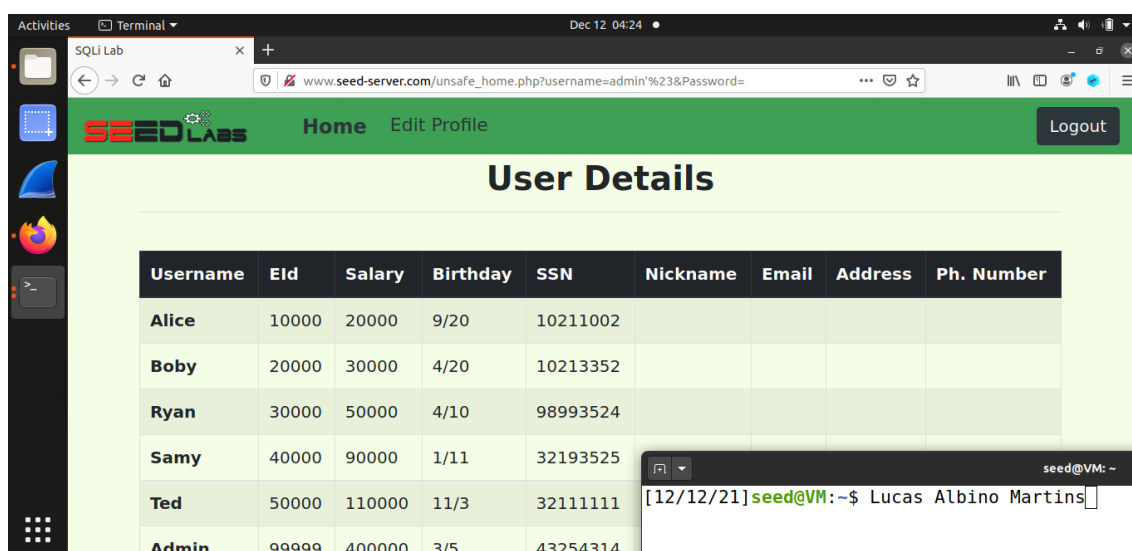


Imagem 10 – Acesso ao user details.

Dessa forma, quando a query for executada, o símbolo # é utilizado para comentar o código, portanto o restante da query terá sido omitida, portanto ao invés de termos algo como “*WHERE name=’admin’#’ and Password=’\$hashed_pwd’*” “. Teremos então a seguinte query sendo executada “*WHERE name=’admin’*”, logo concluímos que a execução da injeção foi realizada com sucesso.

Task 2.2: SQL Injection utilizando a linha de comando:

Agora nessa task utilizaremos os recursos de desenvolvedor que o browser Mozilla Firefox nos fornece. Criando uma chamada HTTP GET com a mesma ideia da Task 2.1, porém foi necessário trocar os símbolos (') e (#) pelo equivalente em URL encoding, que são %27 e %23 respectivamente.

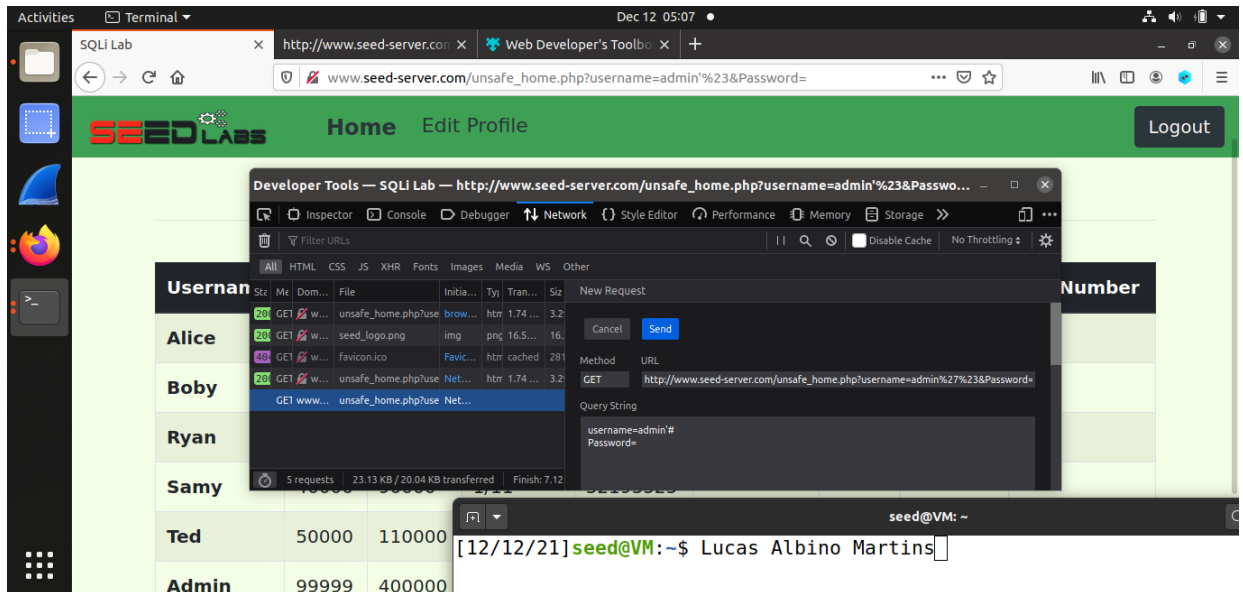


Imagem 11 – Aplicação de linha de comando no painel de desenvolvedor Firefox.

Logo obtemos os mesmos acessos que a situação anterior.

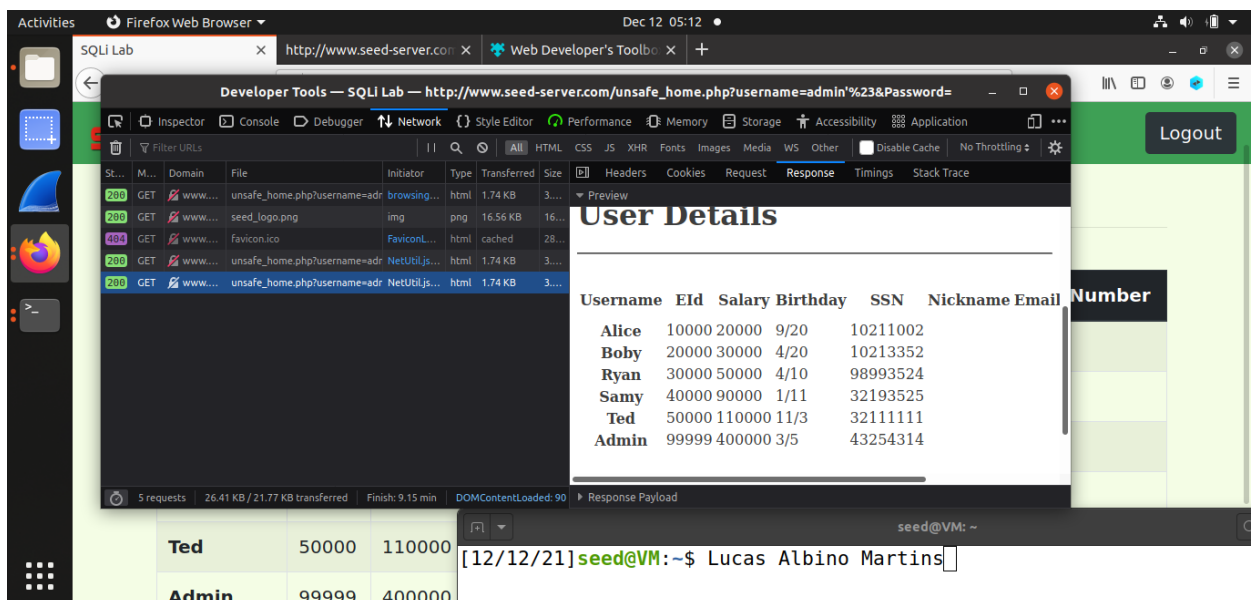


Imagem 12 – Acesso ao User details via Firefox desenvolvedor.

Dessa forma, quando a query for executada, o valor %23 é utilizado para comentar o código, portanto o restante da query terá sido omitida, portanto ao invés de termos algo como “WHERE name=’admin’%27%23’ and Password=’\$hashed_pwd’” “. Teremos

então a seguinte query sendo executada “*WHERE name= 'admin'*”, logo concluímos que a execução da injeção foi realizada com sucesso.

Task 2.3: SQL Injection utilizando APPEND:

Testando agora algo bem semelhante as anteriores, porém iremos adicionar uma query de DELETE antes do símbolo “#” que comentará o restante do query “admin'; DELETE FROM credential WHERE name='Ryan'";#”, porém não houve sucesso ao executar.

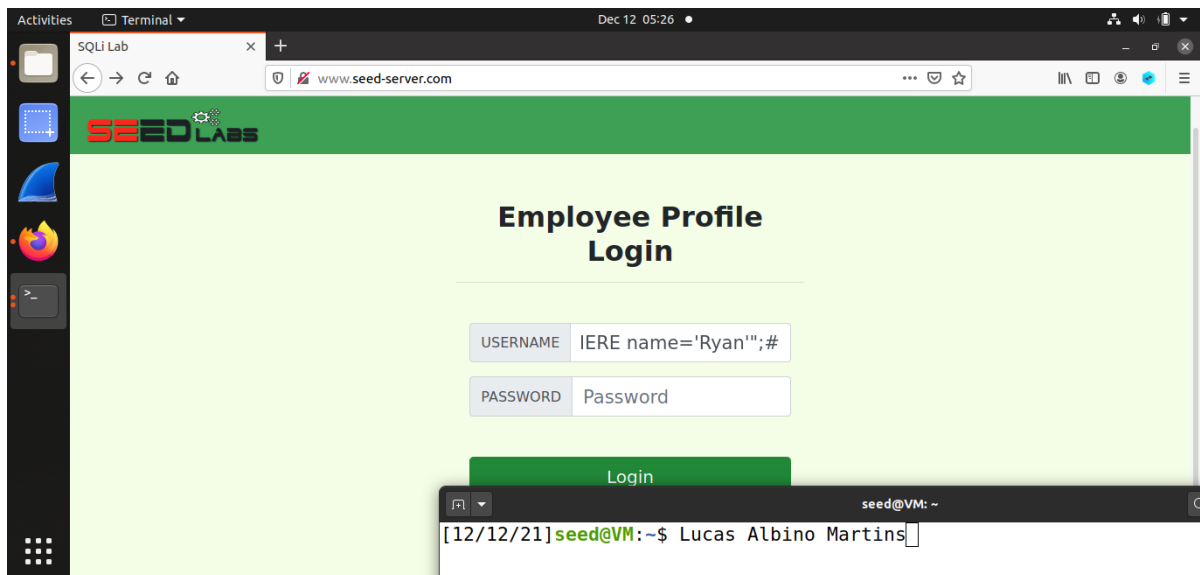


Imagem 13 – Tentativa de APPEND.

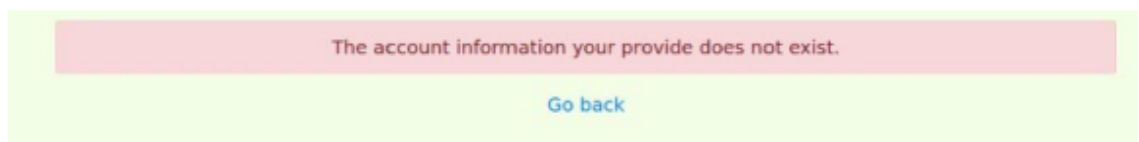


Imagem 14 – Falha na tentativa de APPEND.

A falha no processo ocorreu pois o código utiliza uma extensão do PHP chamada `mysqli::query()` API, que não suporta múltiplas queries serem executadas simultaneamente no banco de dados. Isso significa que não podemos modificar o banco de dados utilizando APPENDS devido ao `mysqli`.

```
// Create a DB connection
$conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error . "\n");
}
return $conn;
}
```

Imagem 15 – Create a DB connection.

Task 3.1: SQL Injection utilizando UPDATE:

Em uma nova tentativa iremos realizar a injeção de SQL diretamente na atualização dos dados dos usuários. Para isso foi necessário logar em um usuário existente Username: Alice, Password: seedalice.

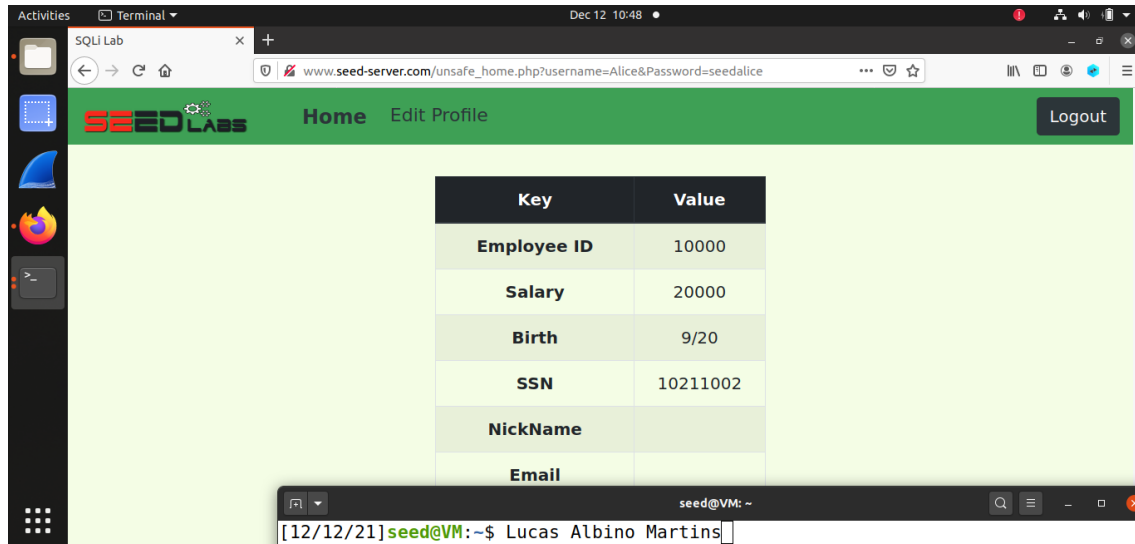


Imagem 16 – User Profile.

Agora aplicando o update, basicamente quando é preenchido os dados para alteração e depois são salvos o arquivo `unsafe_edit_backend.php` fica responsável por atualizar os dados referente ao usuário.

```
$sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
```

Imagem 17 – UPDATE via comando SQL.

Na área de edição do perfil quando digitado o código `"Alice',salary=100000 #"`, na primeira linha aonde se encontra o nome, logo salvo fará com que a query seja executada da seguinte forma:

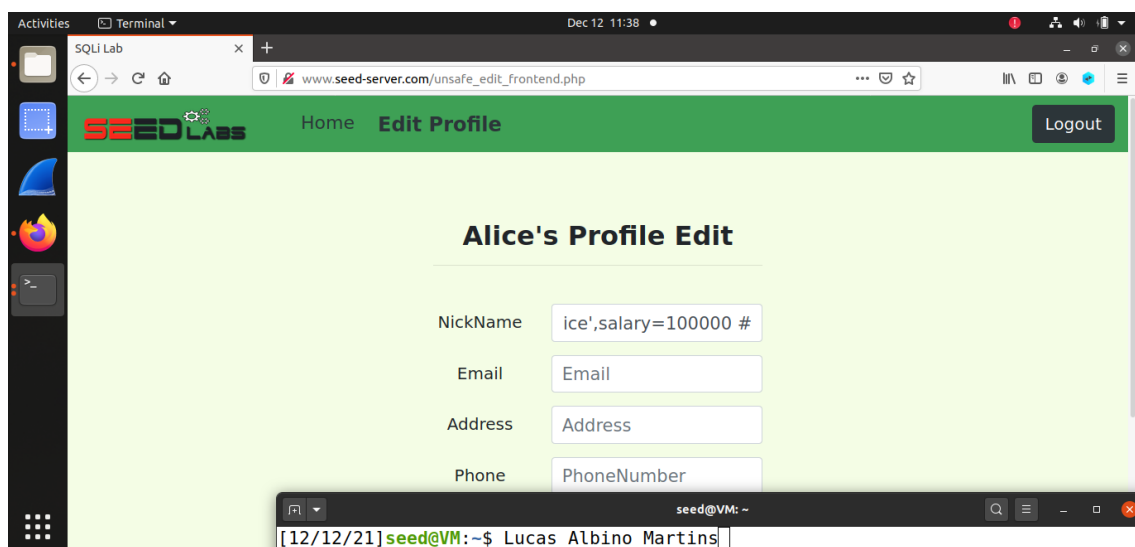


Imagem 17 – Comando para efetuar o Sql injection.

Após a execução a nova tabela de salário irá aparecer:

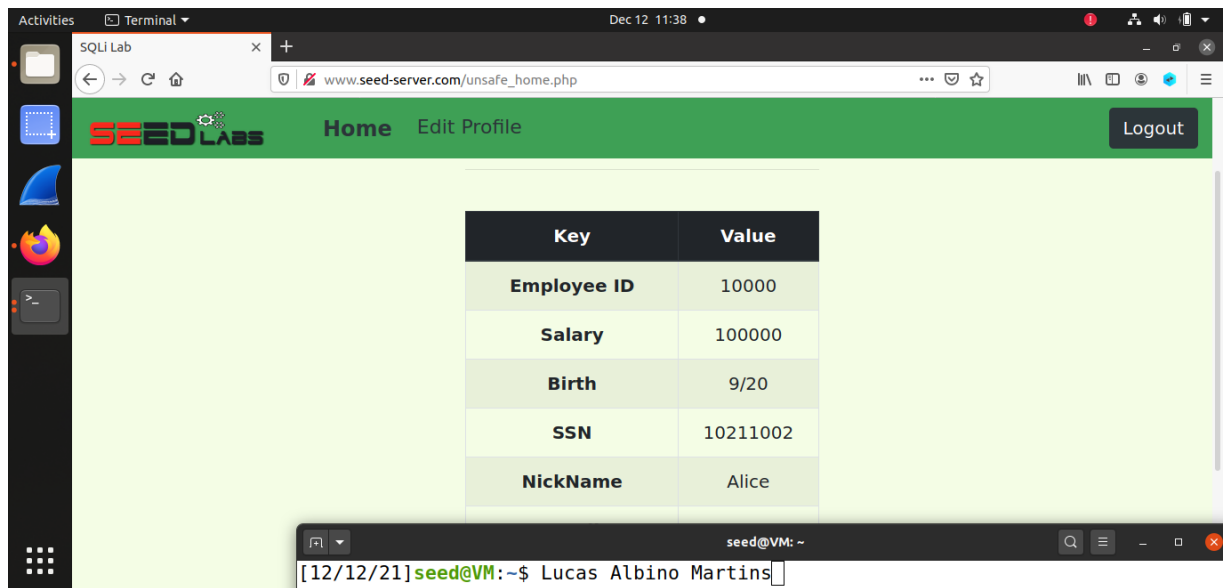


Imagem 18 – User Alice profile.

Logo Alice conseguiu alterar o seu salário por meio de uma injeção de SQL bem sucedida

Task 3.2: Modificando o salário de outra pessoa:

Agora poderemos utilizar uma estratégia semelhante a anterior porem especificando um WHERE no campo de usuário. Primeiro vamos entrar no perfil do Ted para verificar os valores.

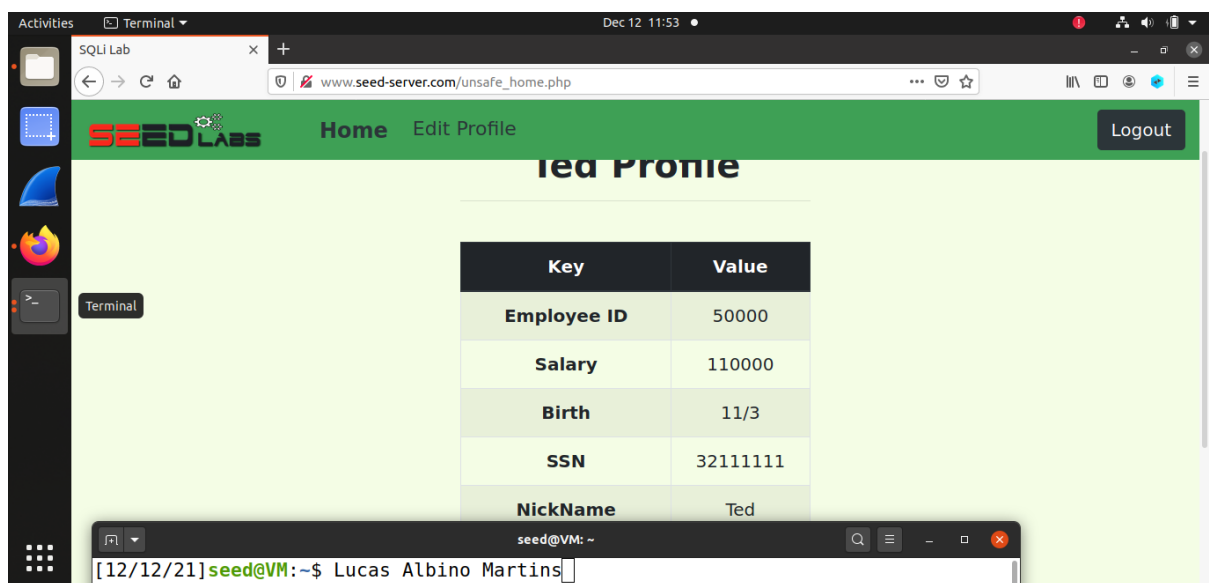


Imagem 19 – Perfil Ted sem alteração.

Alterando o salário do Ted pelo perfil da Alice com o comando “*Alice’,salary=1 where name=’Ted’#*”.

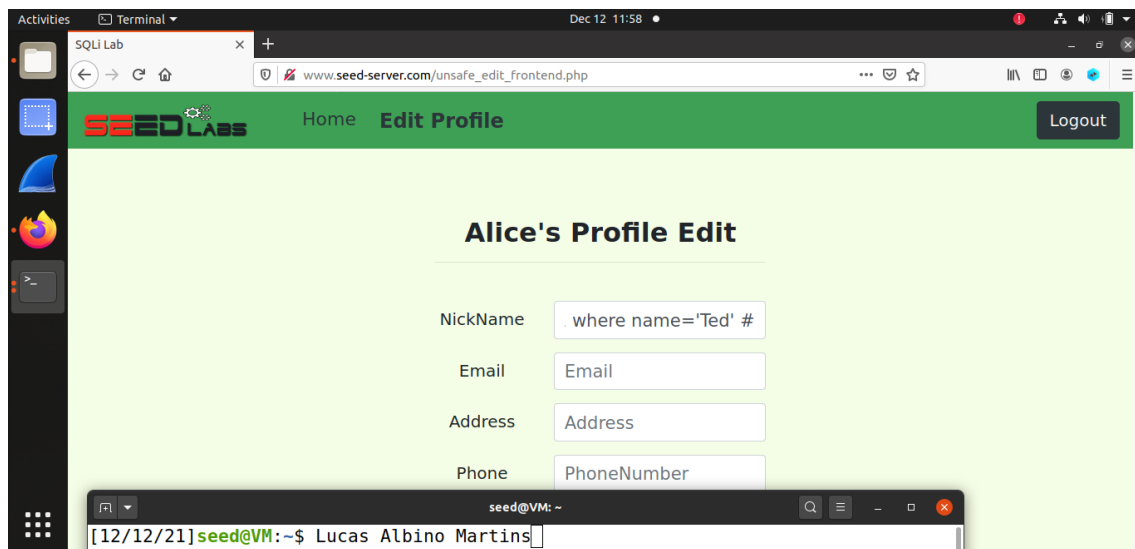


Imagem 20 – Comando Where para execução de SQL-Injection.

Pode ser observado na imagem abaixo o novo valor do salário do TED.

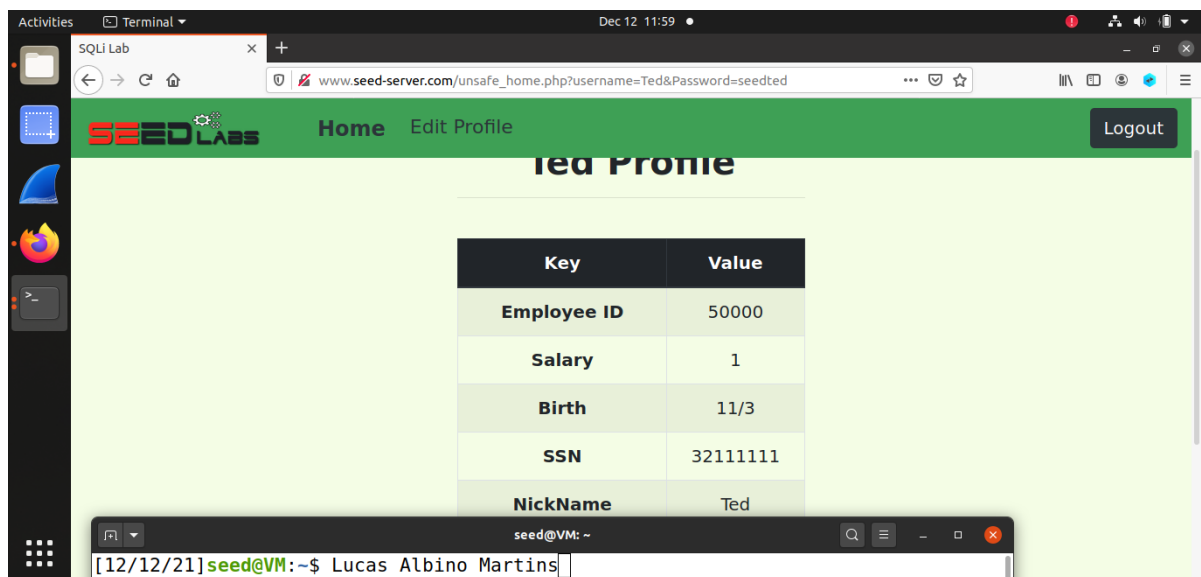


Imagem 21 – Salário do Ted alterado.

Task 3.3: Modificando a senha de outra pessoa:

Para essa estratégia idêntica bem semelhante ao modelo de operação anterior, porém especificando a senha de Ted por exemplo. É necessário frisar que as senhas são encriptadas utilizando a encriptação SHA1, portanto precisamos converter a senha que mudaremos para sha1 antes de envia-las. Para vamos utilizar uma ferramenta online do site website: <http://www.sha1-online.com/>.

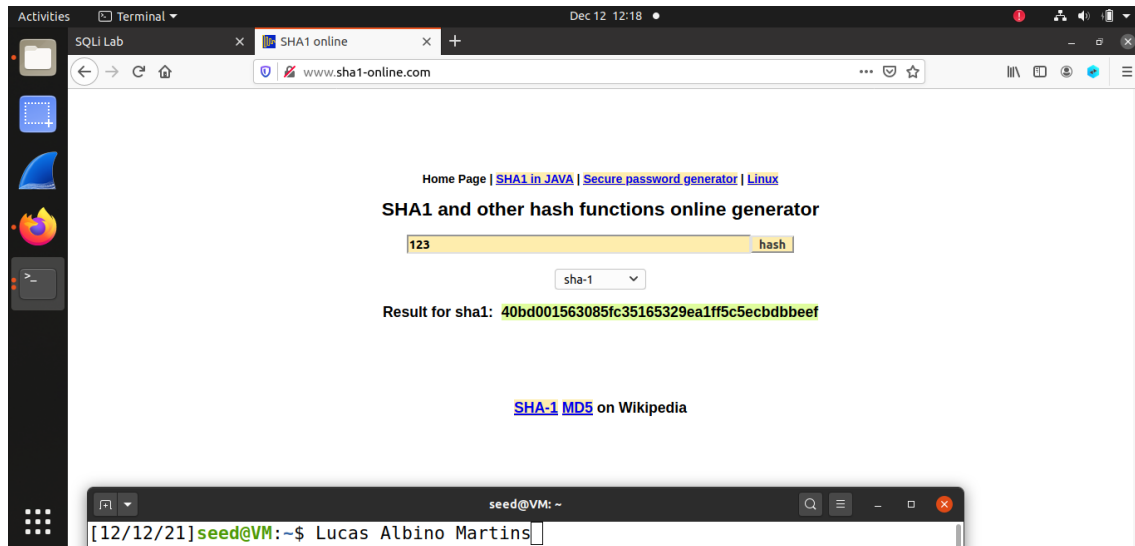


Imagem 22 – Criptografando a senha 123 em SHA-1 MDS.

Agora utilizando o comando parecido com o Where anterior apenas alterando salary para password “Alice',password='40bd001563085fc35165329ea1ff5c5ecbdbbbee' where name='Ted' #” temos uma nova injeção e SQL.

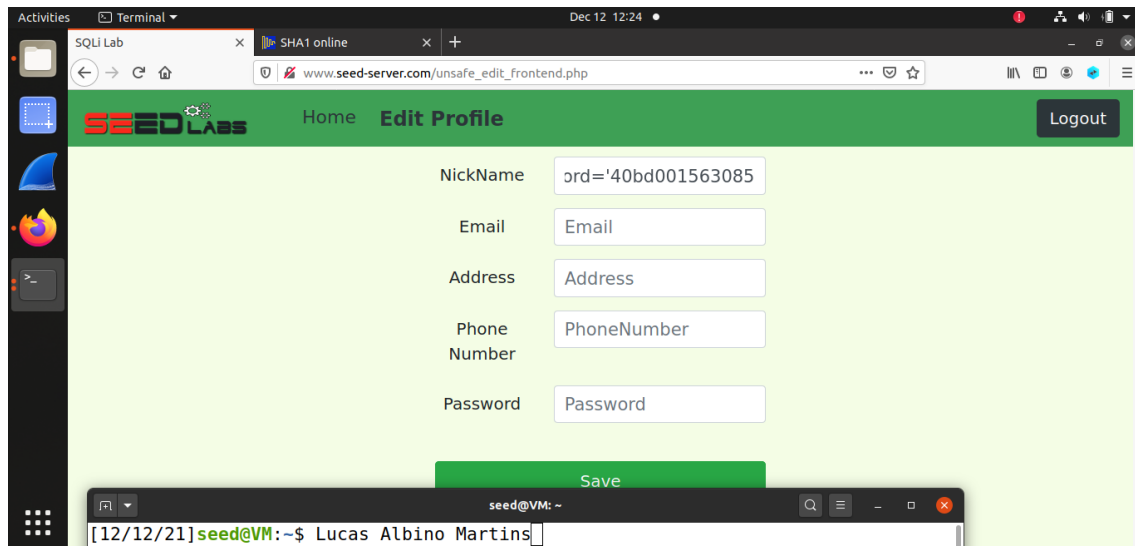


Imagem 23 – Utilizando execução Where para SQL-Injection.

Testando a autenticidade da nova senha 123 para o usuário TED.

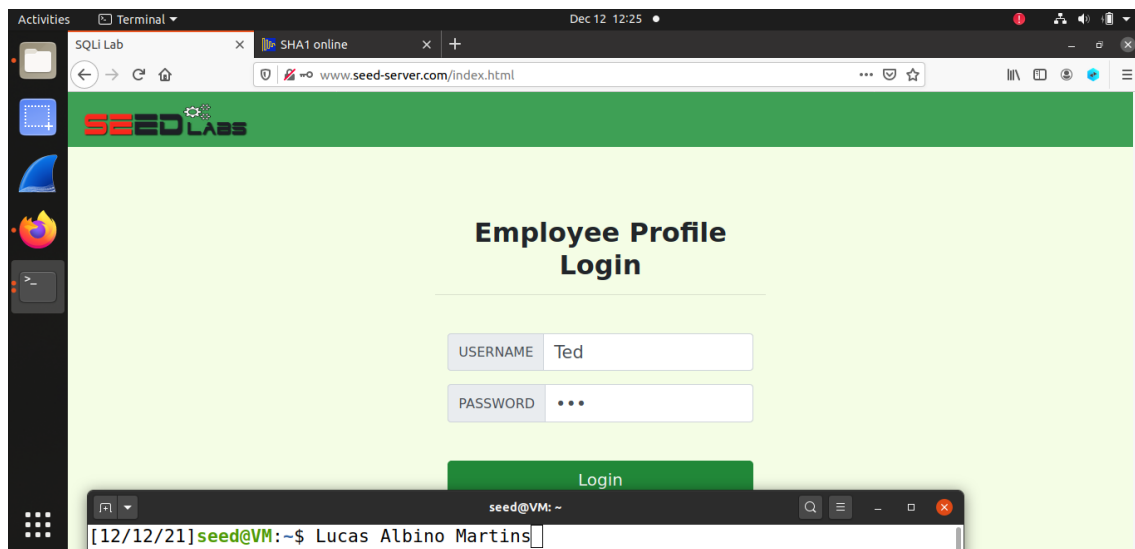


Imagem 24 – Login com a nova senha do usuário Ted.

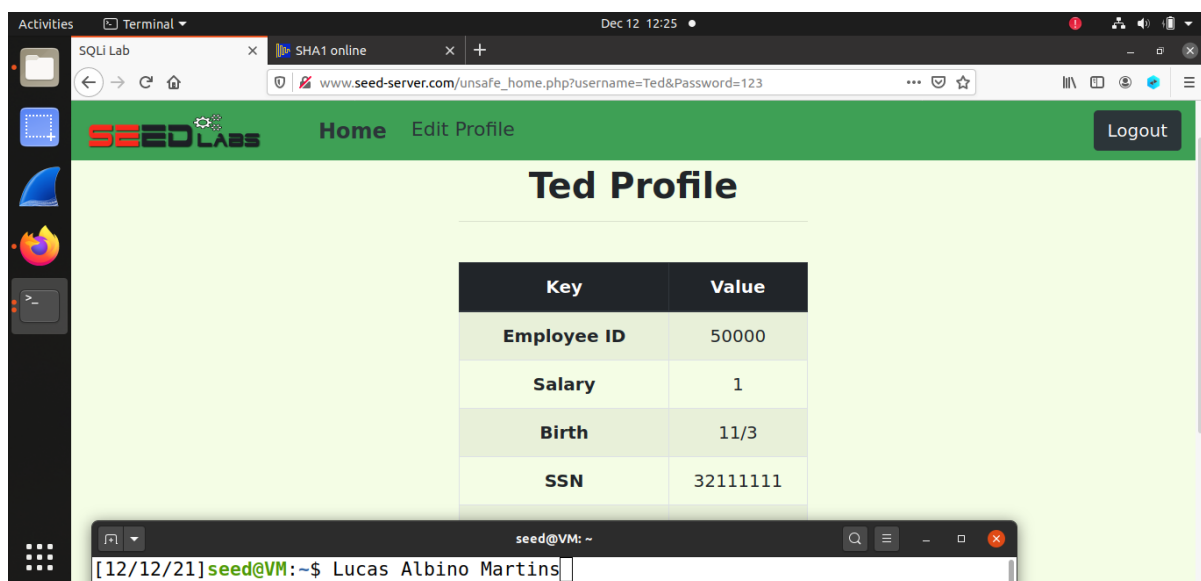


Imagem 25 – Ted Profile com a nova senha.

Logo Alice conseguiu alterar a senha de TED por meio de uma injeção de SQL bem sucedida.

Task 4: Protegendo o Código contra ataques de Injeção de SQL:

Uma alteração foi necessária, editando os arquivos `unsafe_home.php` e `unsafe_edit_backend.php` seguindo o exemplo mostrado na descrição do laboratório para realizar as modificações necessárias.

Seguindo a documentação do laboratório, essas modificações funcionam da seguinte forma “Usando um mecanismo de instrução preparado, dividimos o processo de envio de uma instrução SQL ao banco de dados em duas etapas. A primeira etapa é enviar apenas a parte do código, ou seja, uma instrução SQL sem os dados reais. Esta é a etapa de preparação. Como podemos ver no trecho de código acima, os dados reais são substituídos por pontos de interrogação (?). Após esta etapa, enviamos os dados para o banco de dados usando `bind param ()`. O banco de dados tratará tudo enviado nesta etapa apenas como dados, não mais como código. Ele vincula os dados aos pontos de interrogação correspondentes da declaração preparada. No método `bind param ()`, o primeiro argumento “`i`” indica os tipos dos parâmetros: “`i`” significa que os dados em `$id` têm o tipo inteiro e “`s`” significa que os dados em `$pwd` têm a string modelo”.

No arquivo `unsafe_home.php` foi editado o seguinte trecho:

```
$sql = "SELECT id, name, eid, salary, birth, ssn, password,
        nickname, email, address, phoneNumber
FROM credential";
if (!$result = $conn->query($sql)) {
    die('There was an error running the query [' . $conn->error . ']\n
    ');
}
$return_arr = array();
while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
}
```

Imagem 26 – SELECT no arquivo `unsafe_home.php`.

Alterado para:

```
$stmt = $conn->prepare(
    "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
    email, nickname, Password FROM credential WHERE name=? AND
    Password=? ");
$stmt->bind_param("ss", $input_username, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $birth, $ssn, $
    phoneNumber, $address, $email, $nickname, $pwd);
$stmt->fetch();
```

Imagem 27 – Novo comando SELECT no arquivo `unsafe_home.php`.

No arquivo `unsafe_edit_backend.php` foi editado o seguinte trecho:

```
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($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 password 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);
```

Imagem 28 –Arquivo `unsafe_edit_backend.php` original.

Alterado para:

```
$conn = getDB()
if($input_pwd!=''){
    $hashed_pwd = sha1($input_pwd);
    $_SESSION['pwd']=$hashed_pwd

    $stmt = $conn->prepare("UPDATE credential set nickname=?, email=?, address=? Password=?, PhoneNumber=? WHERE ID=?");
    $stmt->bind_param("sssssi", $input_nickname, $input_email, $input_address, $input_pwd, $input_phonenumber, $id);
}else{
    $stmt = $conn->prepare("UPDATE credential SET nickname=?, email=?, address=?, PhoneNumber=? WHERE ID=?");
    $stmt->bind_param("sssssi", $input_nickname, $input_email, $input_address, $input_phonenumber, $id);
}

$stmt->execute();
$conn->close();
```

Imagem 29 –Arquivo `unsafe_edit_backend.php` original.

No término da edição dos dois arquivos em ambas modificações nenhuma das formas de injeção de SQL anteriormente realizadas obteve êxito, portanto efetivamente protegemos nossa aplicação contra esse tipo de ataque.

Referências:

Hands-on Labs for Security Education. Disponível em <<https://seedsecuritylabs.org/>>. Acesso 10 de dezembro de 2021.

SQL INJECTION ATTACK. Disponível em:

<https://seedsecuritylabs.org/Labs_20.04/Web/Web_SQL_Injection/>. Acesso em 10 de dezembro de 2021.