SQL Injection

Francesco Fossari 0124002327 Corso di Laurea in Informatica

Cos'è una SQL Injection?

La SQL Injection è un tipo di attacco in cui l'utente malintenzionato sfrutta una vulnerabilità della sicurezza Web per visualizzare dati che normalmente non è capace di visualizzare, tramite delle Query SQL malevole. SQL(Structured Query Language) è un linguaggio utilizzato per comunicare con Database online. Questo è un attacco molto diffuso, a causa di possedere il potere di essere usato contro qualsiasi applicazione Web o sito Web che utilizzi un database basato su SQL. Citando Wikipedia, le prime discussioni pubbliche relative all'SQL injection sono apparse attorno al 1998 con un articolo su Phrack Magazine.

Il problema inizia con il modo in cui i Siti web sono scritti. Spesso, infatti, molti di questi siti sono scritti con PHP, ASP, HTML. Questi linguaggi però non comprendono il SQL, quindi le istruzioni SQL solitamente vengono inserite dentro una stringa e qualsiasi cosa l'utente immetta nelle caselle utente e password viene aggiunta a quella stringa.

Ecco un esempio della stringa:

```
1
2  "SELECT *
3  FROM tblUSERS
4  WHERE UserName ='" + txtUserName + "'" AND Password = '"+password +"'"
```

Se l'utente malintenzionato inserisse un'istruzione SQL all' interno dell' form input di UserName e di Password che sarà sempre vera, come:

```
' OR '1'='1
' OR '1'='1' --
' OR '1'='1 ({
' OR '1'='1' /*
```

Formerà una stringa come la seguente:

```
SELECT * FROM users WHERE name = '' OR '1'='1' -- ´;
```

Che forzerà la selezione di tutti gli attributi della tabella users a causa della continua veridicità del codice inserito, garantendo l'accesso ai contenuti della tabella.

Quali sono le conseguenze di un attacco SQL Injection?

Un attacco riuscito di questo tipo può portare all' accesso non autorizzato ai dati sensibili, ed esporre la vittima a gravi conseguenze, a causa della:

- Possibile diffusione delle informazioni personali
- Accesso totale dell'attaccante al sistema
- Compromissione dell'integrità dei dati
- Creazione di Accessi Remoti tramite Backdoors

Quest' attacco può rivelarsi molto dannoso se effettuato ad un'azienda, a causa della perdita di fiducia da parte dei clienti e di un calo drastico della reputazione. Un' altra considerazione importante è che, perdere questo tipo di dati e non garantire la loro sicurezza è illegale, grazie ad un'apposita legge sulla protezione dei dati.

Una particolare informazione è, che quest' attacco viene spesso eseguito dai criminali informatici non tanto per ottenere l'accesso alle informazioni riservate, ma perché dopo aver sottratto diverse tipologie di dati, essi li rivendono ai cosiddetti " Data Brokers " per poter finanziare i loro attacchi successivi e autosostenersi.

Quali sono i sintomi di un attacco SQL Injection?

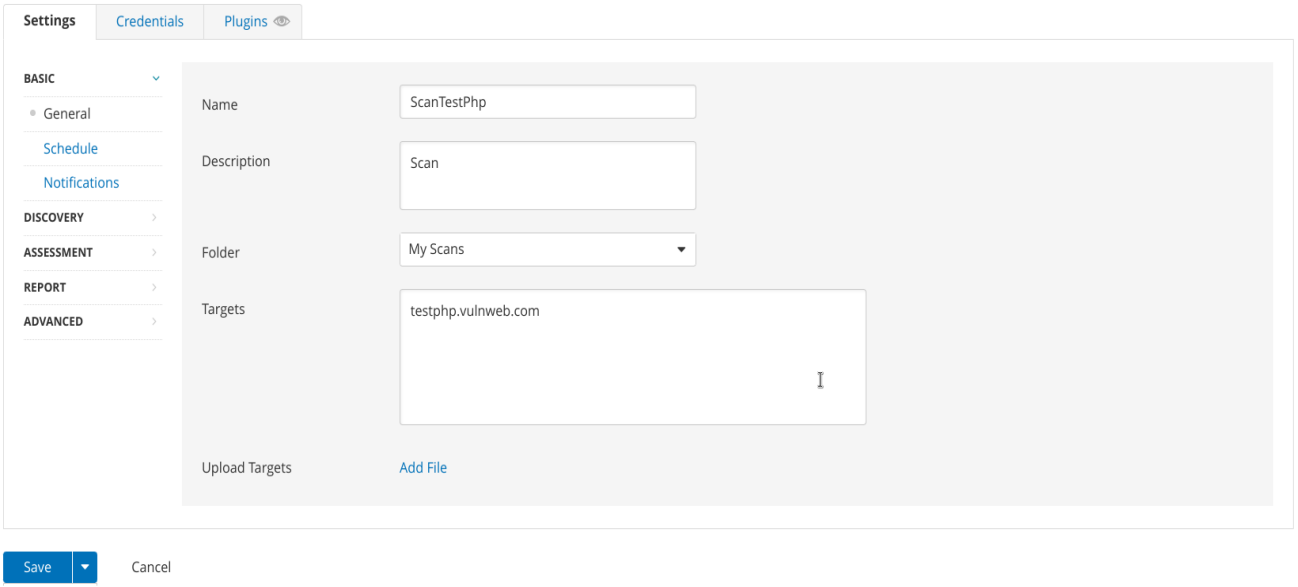
Generalmente è difficile riconoscere un attacco SQL Injection in corso, ma questo non esclude il fatto che questi attacchi sono raramente riconosciuti grazie a diverse anomalie come:

- Numerose richieste in poco tempo
- Messaggi Pop-up sospetti e di errore
- Annunci che reindirizzano a siti Web insoliti

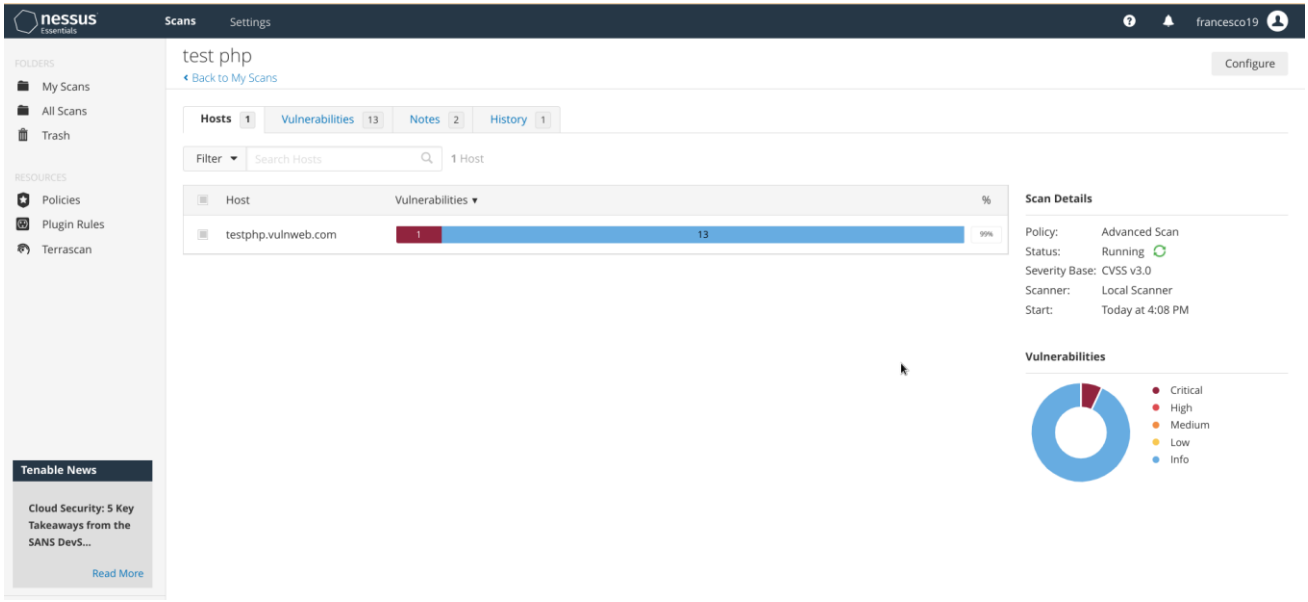
Come possiamo determinare queste vulnerabilità?

Esistono diversi strumenti di scansione, che ci permettono di determinare qualsiasi tipo di vulnerabilità ed il suo livello di criticità. Per riconoscere le vulnerabilità, in questo caso, utilizzeremo Nessus. Questo strumento ci permette di analizzare da remoto una o più macchine e determinare se presentano o meno vulnerabilità. Seguirà una scansione effettuata con Nessus sul sito web http://testphp.vulnweb.com/ , per poter effettivamente constatare il successo della scansione da parte dello strumento e determinare se un attacco di tipo SQL Injection è eseguibile.

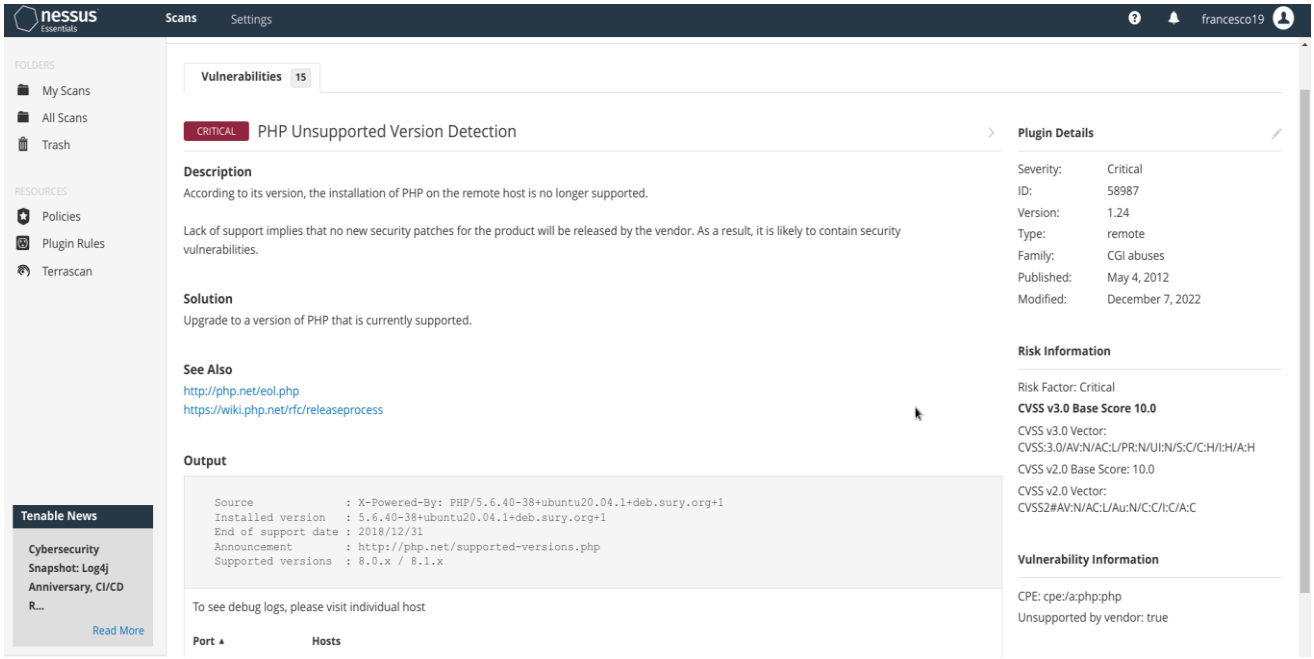
- Inseriamo le informazioni che il tool Nessus ci richiede per effettuare la scansione.



- Avviamo la scansione e attendiamo che essa si completi per poter, in seguito, visualizzare il report finale.



- Una volta terminata la scansione procediamo con l'analisi del report.

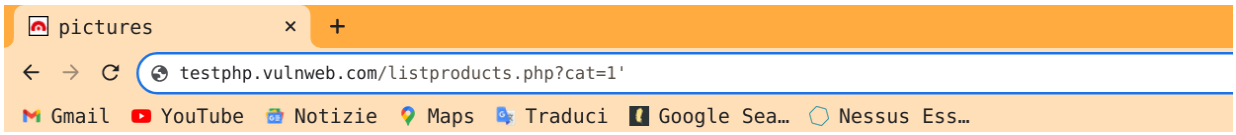


Noteremo che Nessus , alla fine della scansione, ci avviserà di questa particolare vulnerabilità considerata critica. In particolare, il tool ci mostrerà che l' host remoto contiene una versione non supportata di un linguaggio di scripting per applicazioni web.

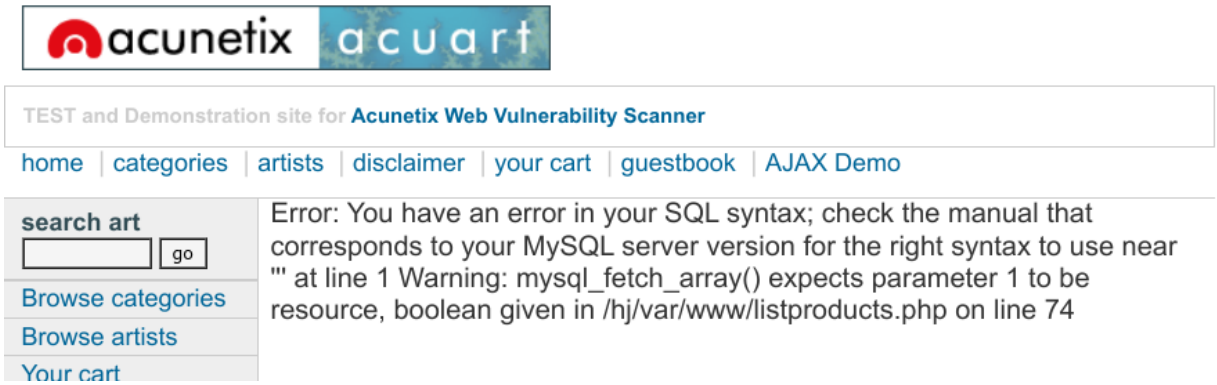
Purtroppo, se non corretta in tempo, l'utilizzo di questa vecchia versione di PHP permetterà ad utenti malintenzionati di sfruttare questa vulnerabilità e di eseguire un attacco SQL injection.

METODOLOGIA SECONDARIA

Un metodo più semplice e veloce per verificare se un sito web è vulnerabile è quello di trovare un parametro URL ed inserire una virgoletta alla fine. Premendo su Categories e poi su Posters, otterremo un parametro URL sul quale effettuare quest' inserimento, esattamente nel modo che segue:



Dopo aver inserito la virgoletta singola, premiamo il tasto Enter sulla nostra tastiera e verifichiamo in quale modo il sito risponde :



Grazie a quest' errore che il sito stesso ci comunica, abbiamo la conferma che esso è probabilmente vulnerabile ad attacchi di tipo SQL Injection.

Come possiamo effettuare un attacco SQL Injection?

A prescindere dal risultato dell'attacco, esistono diverse tecniche che permettono di effettuare una violazione del Database e quindi sottrarre i dati sensibili degli utenti.

In entrambe le tipologie di attacco andremo ad utilizzare il sito Acuart VulnWeb di Acunetix.com (http://testphp.vulnweb.com/index.php/), creato appositamente da Acunetix per testare e far testare diverse tecniche d' attacco senza il bisogno di alcuna autorizzazione, e soprattutto per poter implementare le proprie abilità da Penetration Tester .

ATTACCO TRAMITE STRUMENTI AUTOMATIZZATI

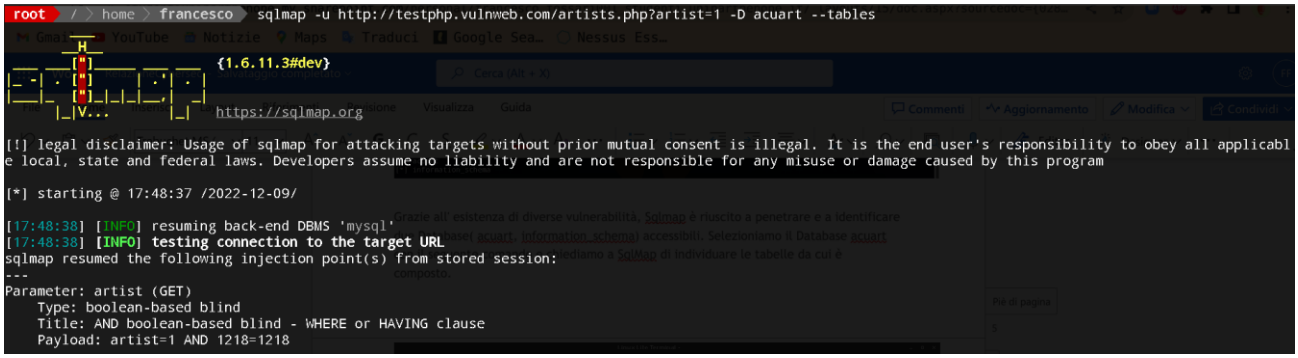
Adesso useremo il tool SQLMap, il quale è uno strumento opensource di Penetration Testing, che permette di eseguire analisi e attacchi automatizzati tramite tecniche di SQL Injection.

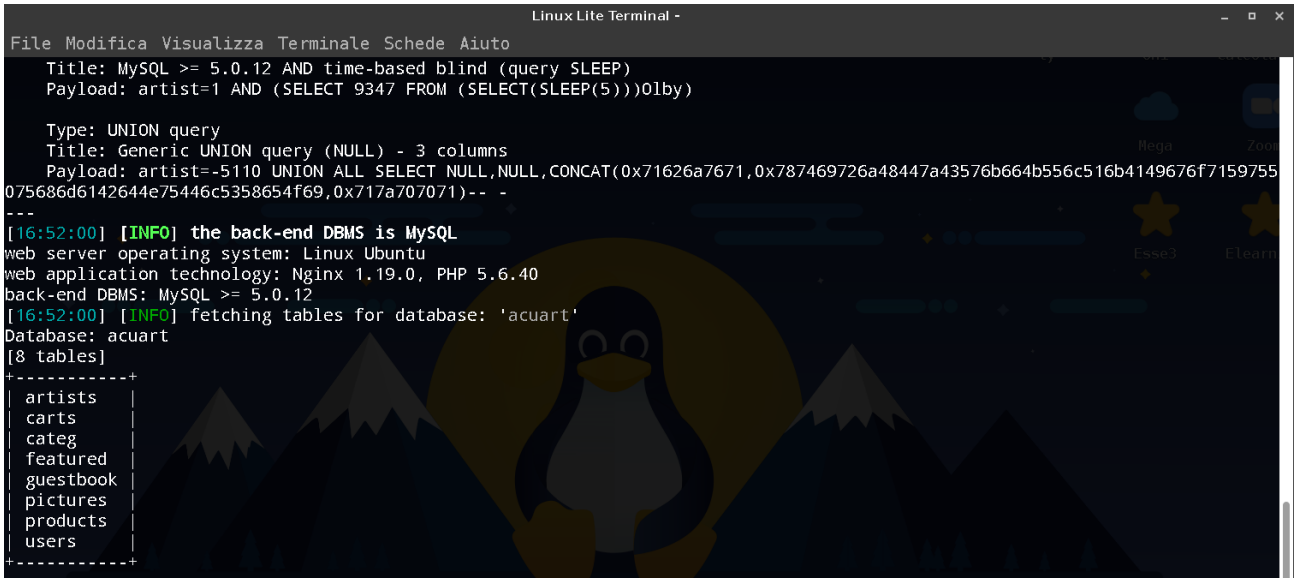
Questo strumento, scritto in Python, viene messo a disposizione su GitHub all' indirizzo https://github.com/sqlmapproject/sqlmap/. Nel caso si utilizzasse Kali Linux come OS, questo strumento è già installato e basterà semplicemente scrivere il comando sqlmap sul terminale per poter accedervi. Dopo aver installato il tool, possiamo procedere con l'attacco.

Questo è il sito web che stiamo per attaccare, troviamo e copiamo il parametro URL che in seguito passeremo a SqlMap.

Apriamo il terminale e digitiamo sqlmap, seguito dall' URL del sito da attaccare e aggiungendo il comando " --dbs ", che ci consente di ottenere la lista dei Database, sempre se il sito web risulta penetrabile.

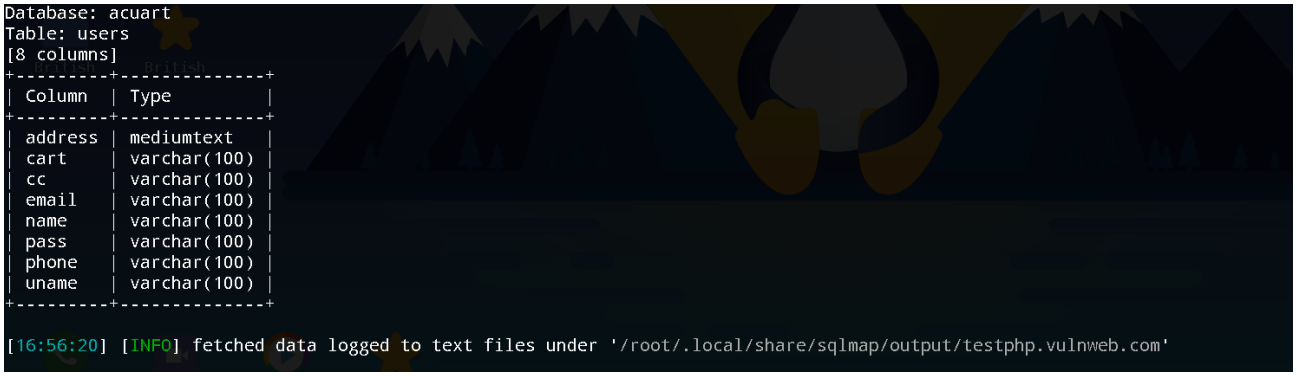
Grazie all' esistenza di diverse vulnerabilità, SqlMap è riuscito a penetrare e a identificare due Database accessibili, cioè Acuart e Information_Schema. Selezioniamo il Database acuart con il seguente comando e chiediamo a SqlMap di individuare le tabelle da cui esso è composto.





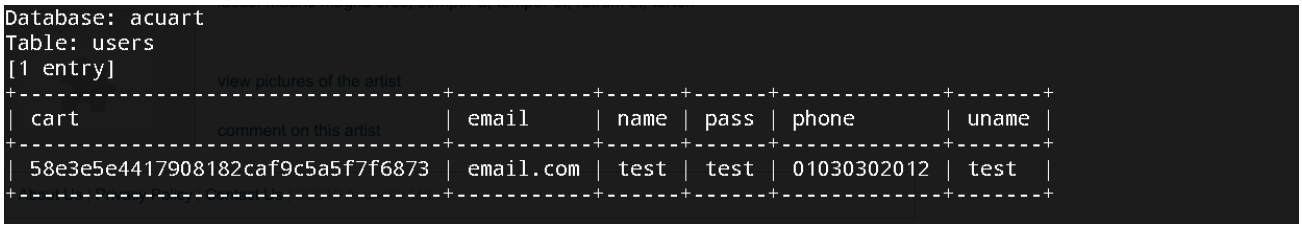
Dopo essere riusciti a risalire alle otto tabelle che compongono il database acuart, selezioniamo quella che più ci interessa, in questo caso Users, e procediamo con l'attacco eseguendo quest' altro comando, in cui richiediamo al tool quali sono gli attributi che compongono la tabella Users.

Dopo aver eseguito l'ultimo comando, SqlMap ci mostra il contenuto della tabella Users come da noi richiesto. Adesso sta a noi la scelta degli attributi presenti da voler estrapolare.



Ora che abbiamo individuato la tabella giusta e conosciamo tutto il suo contenuto, possiamo procedere all' estrapolazione di tutti gli attributi tramite il seguente comando, che selezionerà solo i 6 attributi che abbiamo scelto come cart, email, name, pass, phone e uname. Useremo il comando "--dump" per estrarre tutti gli elementi dalla tabella selezionata.

Arrivati alla fine e avendo eseguito l'ultimo comando, SqlMap finalmente ci mostrerà tutti gli attributi della tabella users da noi richiesti.



Possiamo affermare che l'attacco ha avuto successo al 100% e che i dati sensibili sono stati violati senza particolari problemi o complicazioni.

ATTACCO SENZA STRUMENTI AUTOMATIZZATI

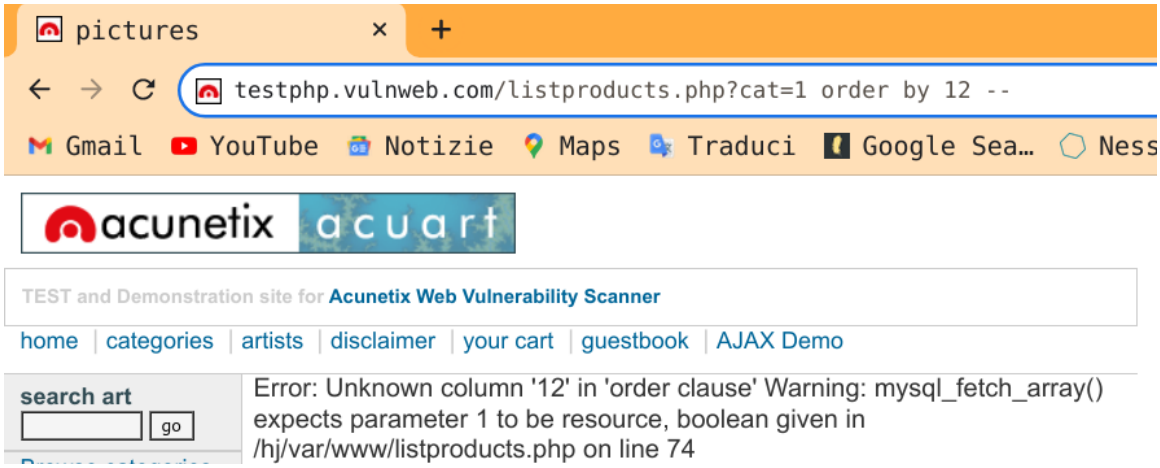
In questa modalità d'attacco non useremo strumenti automatizzati come abbiamo visto in precedenza con SqlMap, ma utilizzeremo una tecnica chiamata SQL Injection Error-Based, che si basa sui messaggi di errore lanciati dal server del database in modo che l'utente malintenzionato possa ottenere informazioni sulla struttura del database. Spesso questa tecnica è sufficiente per consentire un'enumerazione di un intero database. Inoltre, abbiamo in precedenza verificato la vulnerabilità del sito inserendo la virgoletta al termine dell'URL. Quindi possiamo direttamente procedere iniziando subito con l'attacco.

Procederemo seguendo i vari step:

- ## Primo Step

Dopo aver verificato la possibile vulnerabilità, dobbiamo trovare il numero di colonne che costituiscono il Database, utilizzando la Query "Order By".

Scriveremo numeri come 5,10,15,20, ma in questo passaggio mostrerò solamente il numero di colonne già individuato per evitare l'utilizzo di troppe immagini.

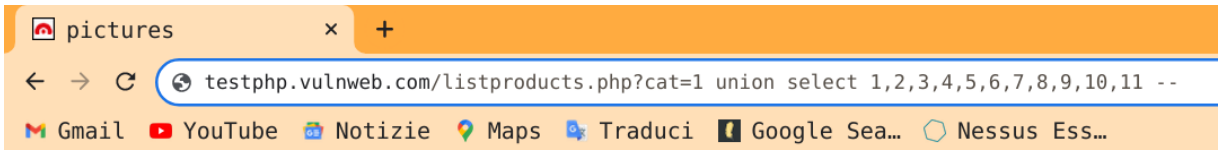


Digitando la query precedente il sito web ci ha risposto con un errore, se al posto di utilizzare il numero 12 utilizzassimo il numero 11, il sito web risponderebbe con nessun errore, dandoci il modo di individuare il numero di colonne esatto, in questo caso 11.

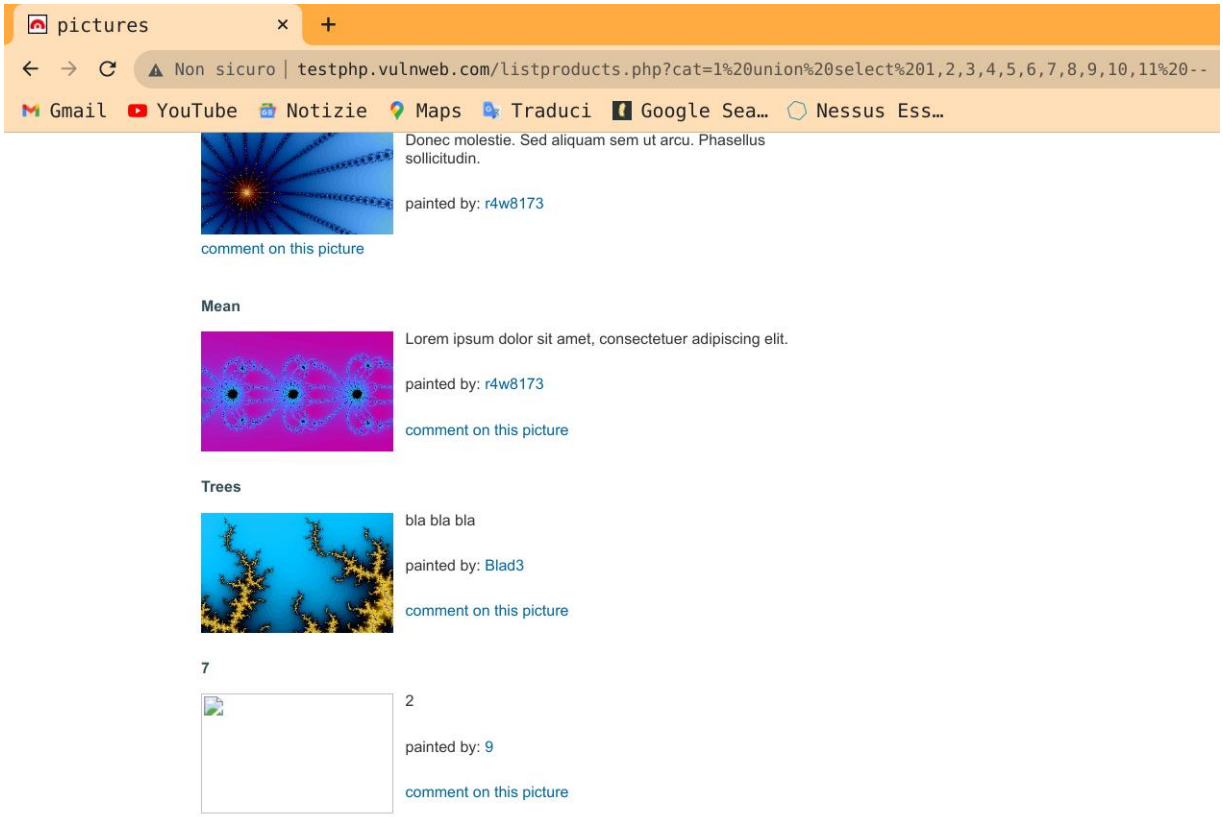
- ## Secondo Step

Dopo aver trovato il numero di colonne, dobbiamo trovare quali sono quelle vulnerabili. Useremo un'altra Query per la ricerca delle colonne vulnerabili:

"Union Select 1,2,3,4,5,6,7,8,9,10,11 -- "



Abbiamo inserito la query, vediamo come il sito risponde:

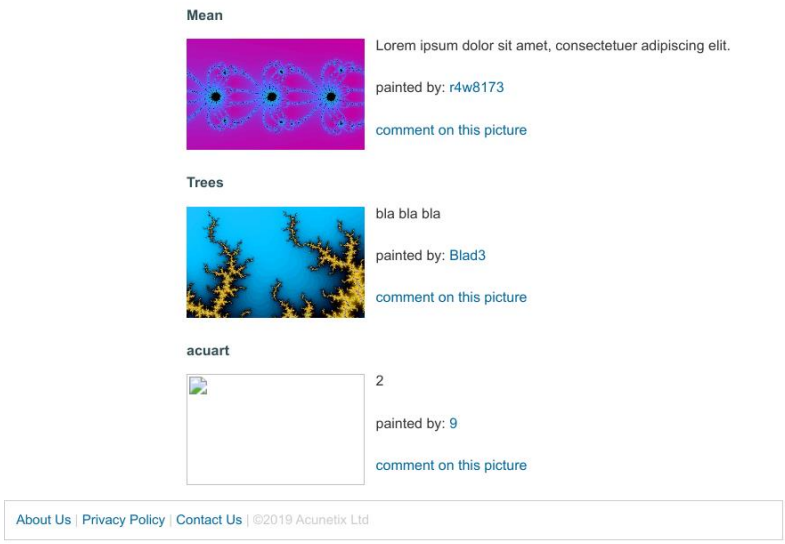


La risposta del sito web, come possiamo notare nell'ultimo riquadro in basso, è il numero appartenente alle colonne vulnerabili, nel nostro caso sono 2, 7 e 9.

- ## Terzo Step

Dopo aver trovato le colonne vulnerabili, possiamo effettuare l'attacco sostituendo alla colonna vulnerabile la Query database() per richiedere il nome del database. Noi utilizzeremo per il nostro scopo la colonna numero sette.

Dopo aver inserito database() al posto della colonna numero sette osserviamo il risultato:

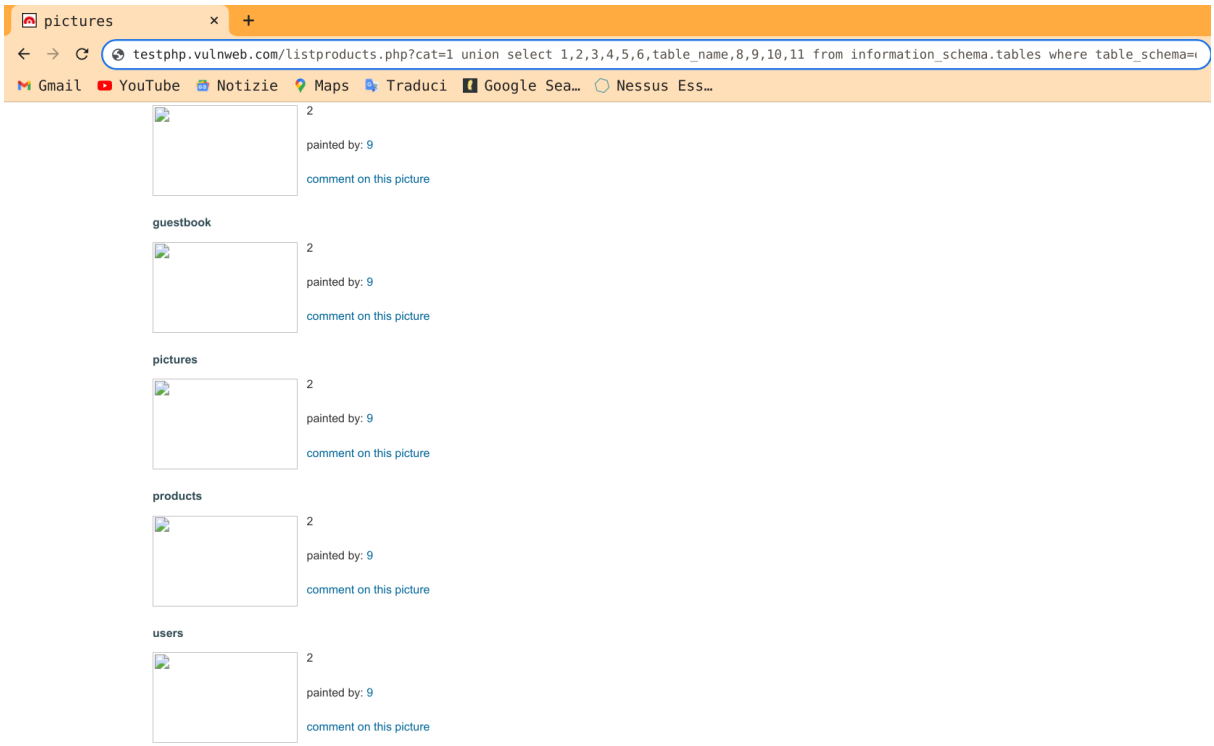


Sempre visualizzando l'ultimo riquadro noteremo il nome acuart, questo significa che la Query database() ha funzionato e che il sito ci ha fornito il nome del database.

- ## Quarto Step

Per ottenere il nome delle Tabelle che lo compongono, dobbiamo utilizzare la prossima Query:

" union select 1,2,3,4,5,6,table_name,8,9,10,11 from information_schema.tables where table_schema=database()-- "

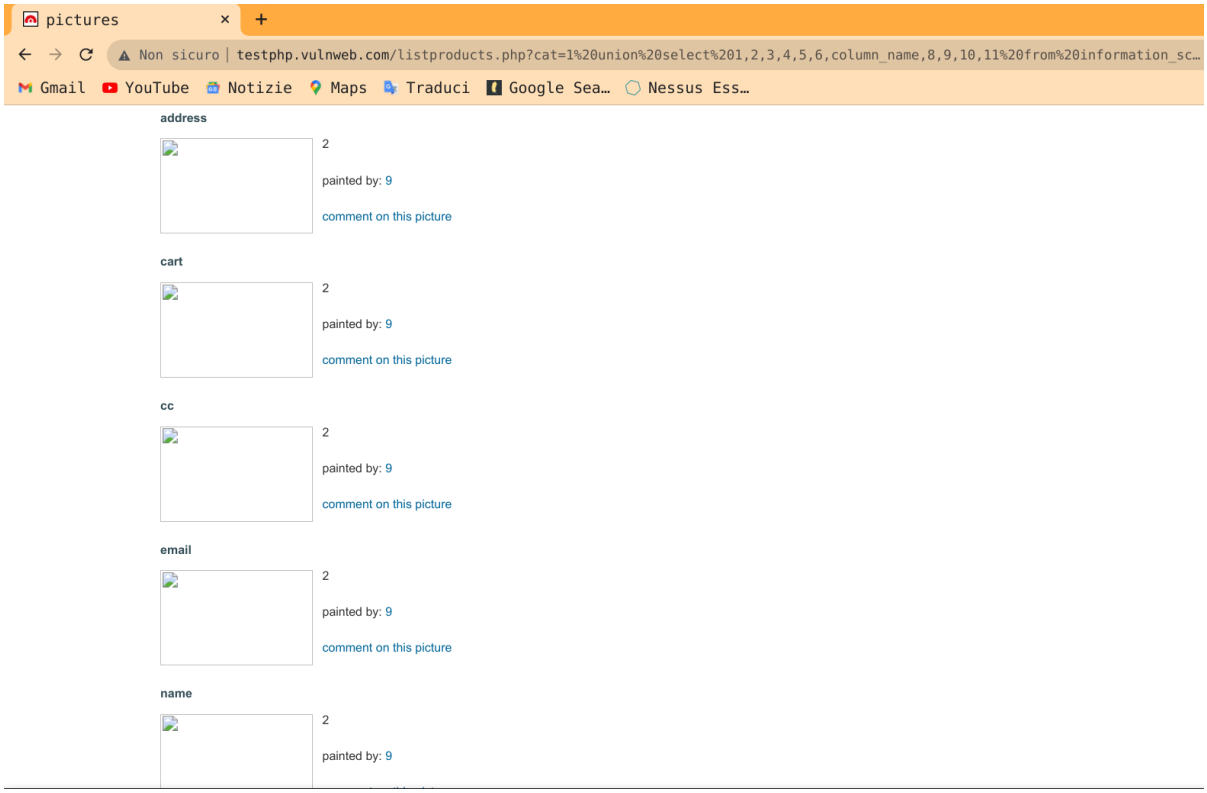


Appureremo che questa lunga Query ci ha fornito un insieme di riquadri, essi non sono nient'altro che tutte le tabelle, con i loro nomi, che compongono il database. La nostra Query ha avuto esito positivo e possiamo procedere con il prossimo step.

- ## Quinto Step

Per ottenere il nome specifico degli attributi di una tabella da noi specificata, dobbiamo eseguire la seguente Query:

" union select 1,2,3,4,5,6,column_name,8,9,10,11 from information_schema.columns where table_name='users'--"

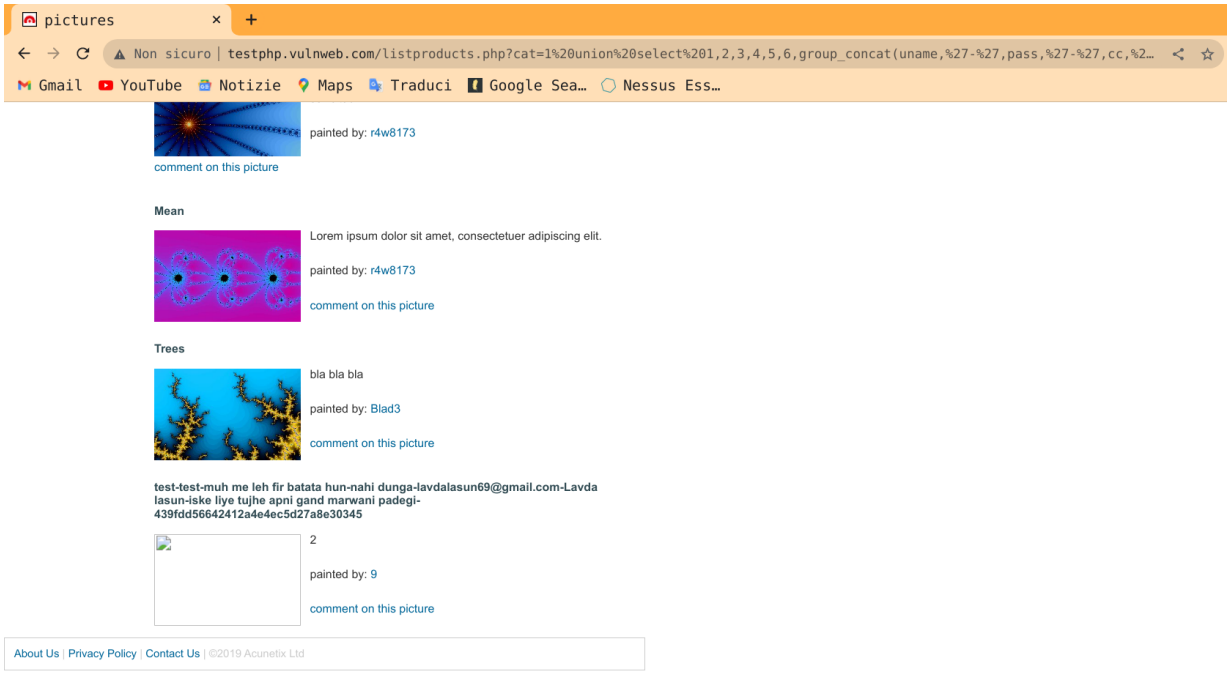


Abbiamo voluto conoscere tramite l'ultima Query, tutti gli attributi che compongono la tabella users. Il sito web ci ha risposto fornendoci i loro nomi, e adesso possiamo procedere con l'attacco ai dati sensibili.

- ## Sesto Step

Quindi, per ottenere l'insieme dei dati personali situati all'interno della tabella users, dobbiamo avanzare con l'utilizzo della Query:

"union select 1,2,3,4,5,6,group_concat(uname,'-',pass,'-',cc,'-',address,'-',email,'-',name,'-',phone,'-',cart),8,9,10,11 from users --"



Ecco qui che l'attacco è andato a buon fine, ottenendo i dati personali dell'utente e permettendoci l'esfiltrazione degli stessi.

Come possiamo prevenire questi attacchi e difenderci?

Poiché questi attacchi sono molto comuni e soprattutto molto datati, nel corso del tempo sono state implementate diverse strategie e diversi strumenti di attacco che ci permettono di violare con poco sforzo un Database poco protetto, permettendo ai dati sensibili di essere violati da utenti non autorizzati e malintenzionati.

Fortunatamente sono state implementate con il passare del tempo, diverse strategie che consentono di proteggere un Database e tutti i dati che esso contiene garantendo la triade CIA. I principali controlli da implementare riguardano principalmente l'input dell'utente, poiché come visto in precedenza, potrebbe introdurre delle Query malevoli. I controlli da implementare sono i seguenti:

- **STATEMENT PARAMETRIZZATI**

Come abbiamo visto in precedenza, una delle principali cause di SQL Injection è la creazione di Query SQL come stringhe che poi vengono inviate al database per essere eseguite. Questo comportamento, conosciuto come Costruzione Dinamica di Stringhe è una delle cause primarie che espone un'applicazione ad essere vulnerabile ad un attacco SQLi.

Come alternativa più sicura alla costruzione di stringhe dinamiche, i linguaggi di programmazione più moderni e le API di accesso ai database consentono di fornire i parametri di una Query SQL tramite l'utilizzo di placeholders, o con le variabili binds, al posto di lavorare direttamente sull'input dell'utente.

Oracle PL/SQL offre la possibilità di usare Query parametrizzate, inoltre supporta i parametri binding utilizzando i due punti con un indice. Nell' esempio riportato di seguito vedremo PL/SQL con parametri associati per creare uno statement parametrizzato.

```
DECLARE
  username varchar(32);
  password varchar(32);
  result integer;
BEGIN
Execute immediate 'SELECT count(*) FROM users where username=:1 and password=:2' into
result using username, password;
END;
```

- **VALIDAZIONE DELL' INPUT**

Precedentemente abbiamo parlato di come evitare l'uso di SQL dinamici per prevenire un attacco SQL Injection. Inoltre, questo non dovrebbe essere l'unico controllo da utilizzare per prevenire una SQLi, infatti uno dei controlli più potenti se messo bene a punto, è quello di controllare l'input che un'applicazione web riceve. La convalida dell'input è un processo in cui si testa l'input ricevuto dall' applicazione per verificare la sua pericolosità ed accertarsi della sua innocuità.

Esistono due tipi diversi di approccio di Convalida dell'input:

APPROVAZIONE ATTRAVERSO WHITELIST

La convalida attraverso una WhiteList è un metodo in cui si accettano solamente input definiti in precedenza non-dannosi. La verifica della convalida può essere coinvolta a causa del tipo, la lunghezza o la dimensione prevista, l'intervallo numerico o altri standard di formato prima di accettare l'input per ulteriori elaborazioni. Questa tipologia di difesa è la più efficace tra i due approcci di validazione dell'input, ma può essere difficile da implementare negli scenari in cui c'è un input complesso.

Approvazione attraverso Blacklist

La convalida attraverso l'uso di BlackList è un metodo in cui si rifiutano gli input considerati malevoli o dannosi. Quest'approccio è generalmente più debole delle WhiteList a causa della grande lista di caratteri considerati potenzialmente pericolosi, la quale richiede un aggiornamento costante e spesso continua a risultare incompleta. Un metodo comune di implementazione di una BlackList è quello di usare espressioni ricorrenti, con una lista di caratteri o stringhe da non consentire, come nell' esempio seguente:

```
' | % | -- | ; | /\* | \\\* | _ | \[@ | xp_
```

Generalmente, è sempre meglio usare la convalida attraverso l'uso di WhiteList quando possibile. In scenari dove l'implementazione delle WhiteList è impossibile, questo metodo fornisce comunque un controllo parziale.

- **CODIFICA DELL'OUTPUT**

Oltre a convalidare l'input ricevuto dall'applicazione, è spesso necessario anche codificare ciò che viene passato tra diverse parti dell'applicazione. Anche nella situazione in cui viene usata la convalida attraverso l'uso di whitelist, spesso il contenuto potrebbe non essere sicuro da mandare al database, specialmente se deve essere usato con il SQL dinamico. Per esempio, il cognome D'Alterio è legittimo e dovrebbe essere consentito dalla convalida con WhiteList, ma questo cognome potrebbe causare problemi significativi nella situazione in cui quest' input viene usato per generare dinamicamente una Query SQL, come la seguente:

```
String sql = "INSERT INTO nomi VALUES('"+ nome +"','" + cognome + "');"
```

Inoltre, un input malevole all'interno del campo nome, come:

```
','');DROP TABLE nomi--
```

Potrebbe essere usato per alterare il SQL eseguito come il seguente:

```
INSERT INTO nomi VALUES('',''); DROP TABLE nomi--','');
```

Si può prevenire questa situazione attraverso l'uso di statements parametrizzati, come visto in precedenza. Nel momento in cui non è possibile il loro uso, sarà necessario

codificare i dati inviati al database. Quest'approccio ha una limitazione, in quanto è necessario codificare i valori ogni volta che vengono utilizzati in una Query del database. Se in un singolo dato inviato la codifica non è stata effettuata, l'applicazione potrebbe essere vulnerabile ad un attacco SQL Injection.

Come visto negli attacchi precedenti effettuati al sito Acuart di Acunetix.org, il back-end del database attaccato utilizzava MySQL. Adesso implementeremo delle tecniche di codifica per MySQL che aiuteranno il Database Acuart ad essere più protetto e sicuro.

Codifica per MySQL

In un server MySQL, come terminatore di una stringa viene usata la virgoletta singola, per cui è necessario codificare la virgoletta singola quando essa è inclusa in stringhe che saranno parte di un SQL dinamico. In MySQL è possibile farlo sia sostituendo il singolo apice con due apici singoli, oppure inserendo il singolo apice con un backslash(\).

Entrambe le opzioni fanno sì che la virgoletta singola venga trattata come parte della stringa e non come un terminatore di stringa, impedendo di fatto a un utente malintenzionato di poter sfruttare la vulnerabilità e quindi utilizzare un attacco SQL Injection con quella particolare Query.

In PHP, viene messa a disposizione la funzione mysql_real_escape(), che automaticamente anteporrà dei backslash ai caratteri speciali prima di inviare la Query al Database, in modo da renderla sicura e prevenire un eventuale attacco.

```
// Escape special characters, if any
$firstname = $mysqli -> real_escape_string($_POST['firstname']);
$lastname = $mysqli -> real_escape_string($_POST['lastname']);
$age = $mysqli -> real_escape_string($_POST['age']);
```

In questo caso, se inserissimo la stringa D'Alterio nel campo input del form, esso verrebbe segnato come D\'Alterio. Se la conservassimo nel database, questa stringa rimarrà D'Alterio senza causare problemi di terminazione della stringa mentre viene manipolata durante la sua esecuzione.

CONSIDERAZIONI AGGIUNTIVE

È necessario considerare altre misure difensive che ci consentono di proteggere la nostra base di dati da eventuali attacchi sferrati da utenti malintenzionati:

- **Esecuzione costante della scansione per applicazioni Web**

Eseguire tramite uno strumento come Nessus, scansioni costanti per identificare e procedere al ripristino di eventuali vulnerabilità prima che sia troppo tardi.

- **Utilizzare sempre le ultime versioni**

L'utilizzo di software aggiornati permette di usufruire delle versioni in cui le vecchie vulnerabilità sono state riconfigurate e corrette, in modo da massimizzare la protezione di ogni dispositivo.

- **UTILIZZARE UN FIREWALL PER APPLICAZIONI WEB (WAF)**

Il Web Application Firewall è un dispositivo di difesa che utilizza un insieme di regole per bloccare il traffico che esce ed entra da un server Web, per provare a prevenire un attacco. Le regole del WAF possono essere personalizzate per proteggere un'applicazione Web da un attacco cross-site scripting(XSS), SQL Injection, Cross-Site Request Forgery(CSRF).

- **UTILIZZARE I PERMESSI DEL DATABASE**

Assegnare ad ogni utente solo i permessi necessari, per tentare di ridurre l'efficacia di un attacco SQL Injection, mirato a dei bug nell'applicazione.

- **USO DI UN SISTEMA PER L'IDENTIFICAZIONE DI INTRUSIONI(IDS)**

Un IDS è un software che viene eseguito su un singolo dispositivo o su dispositivi multipli collegati ad una rete, per monitorare e tracciare la loro attività sulla rete. Esso può essere configurato per valutare i log di sistema, per osservare un'attività sospetta, e disconnettere le sessioni che vengono considerate pericolose o malevoli.

FONTI UTILIZZATE

Di seguito sono riportati tutti i link delle fonti utilizzate per la stesura della relazione.

- CompTIA Security+ Study Guide
- https://Kaspersky.it/resource-center/definitions/sql-injection
- https://it.wikipedia.org/wiki/SQL_injection