Introduzione alle query injection

In questo articolo faremo una puntatina nel mondo delle arti oscure per vedere un tema piuttosto complesso ma che cercherò di illustrare con il massimo della semplicità ovvero le **query injection**. L'argomento è destinato a chi sta sviluppando le sue prime applicazioni web con database e non è ancora completamente in chiaro su quali possano essere le **minacce per la sicurezza**, ma anche per chi, pur essendo più esperto, vuole darsi una rinfrescata.

Origine delle query injection

Questo tipo di exploit parte dal presupposto che se abbiamo un input di dati (un modulo di ricerca o di login ad esempio o anche la querystring di un URL), e questi dati andranno poi a costituire una parte di una query che verrà eseguita sul database, allora è possibile **manipolare la query stessa con risultati che possono essere disastrosi**.
Vediamo come.

Esempio di query injection

L'esempio più semplice ed anche più spaventoso (e quindi vale questo per tutti) è quello che ci viene mostrato da una procedura di autenticazione nella quale non abbiamo tenuto conto della sicurezza.
Come avviene normalmente questa procedura?

- Vengono richiesti username e password tramite un semplice form
- Si esegue una query che andrà a cercare nel database la corrispondenza tra l'username e la password forniti.
- Se questa corrispondenza esiste, la procedura di autenticazione è valida

In codice è qualcosa del genere.

Come vedi, se la query trova un'occorrenza, la procedura di autenticazione andrà a buon fine.
Ora osserva attentamente la query.

Cosa succederebbe se dopo l'AND riuscissimo ad infilare un espressione sempre vera?
Ovvio, avremmo l'autenticazione.
E come poteri fare?
Semplice:

Scriveremo come username: *admin*

E come password: *‘ OR username=’admin*

Vediamo come risulterebbe la query

Questa query andrà cercare la riga dove l’username è *admin* **E** la password è una stringa vuota (sempre falso) **O** l’username è *admin* (sempre vero).
E’ dunque sufficiente che esista un utente *admin* per poterne prendere i privilegi senza conoscerne la password.
Con il primo apice chiudiamo la stringa della password ed inseriamo la nostra stringa maligna.

A questo punto qualcuno potrebbe obiettare che le password sono conservate in forma di hash e dunque, il dato password andrà passato da un algoritmo di hashing rendendo inoffensivo l’attacco.

In effetti con questo codice

La query che ne risulterebbe sarebbe così

Del tutto inoffensiva.

Ma non basta!

Anche in questo caso posso facilmente arrivare ad un exploit.
Passando questi dati:

username: admin’ --
password: *qualsiasi cosa*

Vediamo la query come risulterebbe

In questo caso, dopo avere scritto admin come username chiudo la stringa con il simbolo apice, quindi lascio uno spazio e scrivo due linee (seguite da un altro spazio) che per MySql rappresentano il simbolo del commento. Dunque tutto quello che verrà dopo sarà irrilevante.
In questo modo è sufficiente avere un username valido per accedere all’applicazione in quanto la query che verrà eseguita sul database sarà la seguente:

Combattere le query injection

Come avrai certamente intuito, una delle cose che più di altre permette questo genere di exploit è il carattere apice (ma attenzione non solo).
Dobbiamo quindi renderlo inoffensivo. La prima idea che potrebbe venire è quella di procedere all'escape del carattere apice tramite backslashes (\). **Questa idea appartiene all'archeologia di PHP ed è da considerarsi inefficace**

Il passato che nessuno rimpiange
PHP disponeva addirittura della direttiva *magic_quotes_gpc* che, se attiva, eseguiva l'escape del carattere apice tramite backslashes. Ma questo **non è e non è mai stato un buon metodo** (come non è un buon metodo utilizzare la funzione addslashes()).

- Le strighe salvate nel database con questo metodo vanno poi ripulite in uscita
- *magic_quotes_gpc* è OFF di default da PHP 4.3, deprecata da PHP 5.3 e rimossa in PHP 5.4. Dunque dimenticatela.
- Non garantisce assolutamente la sicurezza. Si veda cosa dice il manuale su questo argomento.

Il vero rimedio
La funzione certamente più importante è mysql_real_escape_string. Il fatto che gli sviluppatori di PHP abbiano sentito la necessità di mettere quel *"real"* nel nome della funzione la dice lunga su altri metodi di escape...
Passare tutte le stringhe destinate a finire in una qualsiasi query ci mette al riparo da **quasi** tutti i problemi. Per incrementare ulteriormente la sicureza possiamo anche:

- Utilizzare espressioni regolari mirate
- Rendere inoffensivo eventuale codice HTML (che può contenere del codice javascript dannoso) con le apposite funzioni (strip_tags o htmlentities)
- Forzare il tipo numerico dove è richiesto

Prepariamo ora una funzione di base per ripulire le stringhe destinate a diventare parte di una query. Inizieremo con il rimuovere eventuali backslashes aggiunte nel caso in cui *magic_quotes_gpc* sia attivato. Per sapere se questa direttiva è attiva, PHP mette a disposizione la funzione get_magic_quotes_gpg che ritorna TRUE se la direttiva è ON, altrimenti false.

Qui però metterei un accorgimento. Se la direttiva *magic_quotes_gpc* è stata rimossa, probabilmente un giorno verrà rimossa anche la funzione *get_magic_quotes_gpc*.
Dunque meglio verificarne l'esistenza prima di utilizzarla, in questo modo:

Ed ora passiamo la stringa per *mysql_real_escape_string* in modo da ottenere l'escape dei caratteri speciali, da *strip_tags* in modo da eliminare i tag html, ed infine facciamo un *trim* che non

fa mai male. Fino ad ottenere questa funzione.

Conclusione

In questo articolo abbiamo trattato **le basi delle query injection** ed abbiamo visto i principali metodi di difesa. Il discorso rimane comunque molto complesso ed esistono degli esempi di exploit che rasentano la follia.
Il consiglio è quello di mai accontentarsi. La funzione appena esposta è una valida base, ma per migliorare la sicurezza considera anche delle espressioni regolari mirate al contenuto atteso.

Un'ultima ed importante nota. **L'utilizzo delle tecniche esposte in questo articolo su siti web che non ti appartengono direttamente, costituisce un reato penale (indipendentemente dall'esito dell'exploit).**