SQL Injection è una tecnica che consente a un attaccante di inserire comandi SQL arbitrari nelle query eseguite da un'applicazione Web sul proprio database. Può funzionare su pagine Web e app vulnerabili che utilizzano un database relazionale.
Un attacco riuscito può comportare l'accesso non autorizzato a informazioni riservate nel database o la modifica di dati. In alcuni casi, una SQL Injection riuscita può arrestare o addirittura eliminare l'intero database.

Come difendersi

Come prima misura, occorre validare l'input, sottoponendolo a rigidi controlli, come già illustrato nei punti precedenti.
Le query SQL non devono mai essere realizzate concatenando stringhe con l'input esterno. Bisogna invece utilizzare componenti di database sicuri come le stored procedure (stored procedures), query parametrizzate alle quali si associano i valori in input.
Una soluzione che può essere d'aiuto consiste nell'utilizzazione di una libreria ORM, come EntityFramework, Hibernate o iBatis.
Occorre limitare l'accesso agli oggetti e alle funzionalità del database, in base al "Principle of Least Privilege" (non fornire agli utenti permessi superiori a quelli strettamente necessari).
Esempio:
Consideriamo la seguente query:

```
SELECT * FROM Tabella WHERE username='$user' AND password='$pass'
```

$user e $pass sono impostate dall'utente e supponiamo che nessun controllo su di esse venga fatto.
Vediamo cosa succede inserendo i seguenti valori:

```
$user = ' or '1' = '1
$pass = ' or '1' = '1
```

La query risultante sarà:

```
SELECT * FROM Tabella WHERE username='' or '1' = '1' AND password='' or '1' = '1'
```

Nell'approccio white list viene proposto un insieme di caratteri validi. Ad ogni richiesta, se l'input ricevuto contiene dei caratteri non presenti in tale lista, allora segnaleremo un errore. Ciò comporta un'attenta definizione della lista in fase di definizione dei requisiti dell'applicazione, oltre che una corretta gestione dei caratteri.
Oltre la white list, si può anche usare il metodo della concatenazione delle variabili con uso della funzionalità "quote".

Esempio:
Forma non corretta:

```
SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" | &UserInput | "'", &Name,
&Phone);
```

Forma corretta

```
SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
Quote(&UserInput) | "'", &Name, &Phone);
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html CWE-89, Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

7.3.4 Ulteriori indicazioni per lo sviluppo sicuro

Di seguito vengono descritte ulteriori direttive per lo sviluppo PL/SQL in sicurezza.

7.3.4.1 Posizionamento delle procedure PL/SQL

È necessario valutare attentamente la posizione in cui si collocano le procedure sviluppate:

- In file separati, organizzati per categoria, sul filesystem del db server:
- Minor numero di vulnerabilità derivanti dal fatto che il codice non viene precaricato
- Implementazione di meccaniche di "failover" più semplice