

- Esempio:
Si consideri la query: \$sql = "SELECT * FROM fatture WHERE nome_cliente LIKE '%" . \$nome . "%' AND ref_cliente=2 ORDER BY num_fattura ASC"
Se il parametro \$nome fosse acquisito da input utente e inizializzato alla stringa: %' #
La query risultante sarebbe: SELECT * FROM fatture WHERE nome_cliente LIKE '%"%' # AND ref_cliente=2 ORDER.

Esempio di **Script vulnerabile a SQL Injection**:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <TRACKING>
- <Anagrafica>
  <TipoRichiesta>TRACKING </TipoRichiesta>
  <CodiceFiscale />
</Anagrafica>
- <Log COD="10">
  <EsitoRichiesta>N</EsitoRichiesta>
  <Tipologiaerrore>1</Tipologiaerrore>
  <Descrizioneerrore>ORA-00920: invalid relational operator</Descrizioneerrore>
  <Log>ORA-00920: invalid relational operator</Log>
</Log>
</TRACKING>
```

Contromisure

Per impedire un attacco di SQL Injection è necessario evitare di concatenare le stringhe delle query e affidarsi alle stored procedures e alle query parametriche (prepared statement). Può essere utile utilizzare una libreria ORM come EntityFramework, Hibernate, or iBatis, ma questa tecnologia – di per sé - non mette al riparo dalla SQL Injection.

6.2 Session Management

Le problematiche di Session Management sono particolarmente comuni nelle applicazioni Web e più in generale in tutte quelle applicazioni che gestiscono sessioni di collegamento individuali di ciascun client. Errori di progettazione del software in questo caso possono consentire a utenti non autorizzati di accedere a dati protetti. Un aggressore può appropriarsi della sessione di collegamento di un utente lecito operando al suo posto, impedendo a quest'ultimo di accedere a una o più risorse.

La prevenzione di tali attacchi può essere messa in atto in diversi modi, ad esempio rigenerando l'id di sessione a ogni login. La stessa cosa può essere fatta con i cookies, rigenerandoli a ogni chiamata. È possibile utilizzare un id di sessione molto lungo, in modo che non possa essere facilmente indovinato. Nessuna di queste misure, tuttavia, riesce a eliminare del tutto il rischio di furto di sessione. L'unico rimedio veramente efficace è utilizzare una connessione sicura con SSL/TLS.

Di seguito sono descritte le principali cause e vulnerabilità che danno origine a problematiche di Session Management.

6.2.1 Session Stealing e Hjhacking

Un aggressore che riesce ad ottenere l'identificativo di una sessione (detto anche token) o il cookie di un utente e replicarlo esattamente in una o più richieste inviate al server, ha la capacità di accedere ad aree o risorse che dovrebbero solo essere riservate all'utenza lecita, bypassando in modo diretto i meccanismi di autenticazione dell'applicazione.

Sono diverse le cause che agevolano o permettono di portare a termine attività di Session Stealing/Session Hjhacking, di seguito vengono proposte le più comuni.

Esempio: