Esempio:
Qui viene aperto un socket con i dati (non validati) dell'utente:

```
var unsafe_socket;
function createSocketToServer_Unsafe() {
     var params = new URLSearchParams(document.location.search);
     var wsurl = params.get("ws_url");

  unsafe_socket = new WebSocket(wsurl);
     unsafe_socket.onopen = function(){
               sendMessage(unsafe_socket);
     }
     unsafe_socket.onmessage = function(msg){
               receiveMessage(unsafe_socket);
     }
}
```

Qui di seguito, invece, la versione sicura:

```
var safe_socket_hc;
function createSocketToServer_SafeHardcoded() {
  safe_socket_hc = new WebSocket(SERVER_WS_URL);
     safe_socket_hc.onopen = function(){
               sendMessage(safe_socket_hc);
     }
     safe_socket_hc.onmessage = function(msg){
               receiveMessage(safe_socket_hc);
     }
}
```

Maggiori informazioni: http://cwe.mitre.org/data/definitions/99.html

7.11.6 Client Second Order Sql Injection

Come riconoscerla

L'applicazione comunica con il suo database inviando una query SQL testuale. L'applicazione crea la query semplicemente concatenando le stringhe con dati ottenuti dal database. Poiché tali dati potrebbero essere stati precedentemente ottenuti dall'input dell'utente e non sono stati verificati né tanto meno bonificati, potrebbero contenere comandi SQL, che potrebbero essere interpretati come tali dal database.

In questo modo, un malintenzionato può accedere direttamente a tutti i dati del sistema. L'aggressore sarebbe in grado di rubare qualsiasi informazione sensibile memorizzata dal sistema (come i dettagli personali dell'utente o le carte di credito) e possibilmente modificare o cancellare i dati esistenti.

Come difendersi

Procedere con la validazione dei dati, prima di salvarli nel database.
Effettuare sempre la validazione dell'input, prima di utilizzarlo all'interno dell'applicazione. Occorre controllare il tipo del dato, la sua dimensione, l'intervallo di validità (range), il formato ed eventuali valori attesi (white list).
Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
Codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web (encoding). La codifica dovrebbe essere sensibile al contesto, in base al tipo di dato che si vuole neutralizzare: se ci si aspetta che possa esserci codice HTML abusivo, occorre codificare gli eventuali tag HTML, se ci si potrebbe trovare di fronte a uno script, allora bisogna codificare gli elementi sintattici di Javascript, ecc.
Invece di concatenare le stringhe:

- Utilizzare componenti di database sicuri come le stored procedure, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).