Il controllo di SQL Injection nelle pagine Php

Con il termine **SQL Injection** si intende l'aggiunta di istruzioni SQL nell'input che l'utente fornisce nel browser tramite un form HTML. Viene utilizzato dagli utenti non autorizzati (*hacker*) per accedere ai database senza controlli e senza identificazione oppure per far eseguire comandi SQL diversi da quelli previsti dall'applicazione. Poiché questo comporta rischi per la sicurezza dei database, è importante prevenire l'inserimento di codice SQL indesiderato (nel gergo informatico, *malicious SQL*, che potremmo tradurre con codice infido o codice doloso).

Per comprendere come funziona l'*SQL Injection*, si consideri la pagina Web che richiede all'utente lo username e la password per accedere a un servizio Internet. Essa contiene due caselle di testo all'interno di un form HTML, denominate *utente* e *password*, e richiama l'esecuzione di una pagina Php che legge dalla tabella *Utenti* di un database l'elenco degli utenti autorizzati, controllando che i dati forniti corrispondano all'identificativo dell'utente. Supponiamo che la tabella *Utenti* contenga, per ogni riga, due soli campi: *username* e *password*.

I seguenti frammenti di codice descrivono le principali operazioni svolte dalla pagina Php.

- Acquisizione dei dati dal form della pagina Web:

```
$user  = $_POST["utente"];
$pwd   = $_POST["password"];
```

- Costruzione del comando SQL:

```
$sqlcmd  = "SELECT * FROM UTENTI ";
$sqlcmd .= "WHERE username = '$user' and ";
$sqlcmd .=       "password ='$pwd'";
```

- Esecuzione del comando SQL e controllo dei dati:

```
$risultato = mysql_query( $sqlcmd );
if ( mysql_num_rows($risultato)==0 )
   echo "Utente non identificato");
else
   echo "Utente identificato");
```

Se l'utente fornisce come *username* 'user1' e come *password* 'passw1', il comando SQL nella pagina Php diventa:

```
SELECT * FROM Utenti
WHERE username = 'user1' and password = 'passw1'
```

Se l'utente esiste nel database, il processo di identificazione fornisce esito positivo.
Supponiamo ora che l'utente inserisca come *password* o per *username* e *password*, la seguente sequenza di caratteri:

```
1' Or '1'='1
```

Il comando SQL costruito in *$sqlcmd* diventa:

```
SELECT * FROM Utenti
WHERE username = '1' Or '1'='1' and password = '1' Or '1' = '1'
```

Le condizioni scritte dopo *Or* sono sicuramente vere ('1' = '1'), rendendo complessivamente vere le condizioni per lo *username* e la *password*: il controllo di identificazione viene superato in modo positivo. In questo modo un utente potrebbe accedere ai dati senza conoscere username e password.
Per impedire l'*SQL Injection* occorre stabilire permessi di accesso più restrittivi per gli utenti del database e inserire all'interno delle pagine Php meccanismi di validazione dei dati forniti dall'utente prima che essi vengano utilizzati per l'accesso alle tabelle del database.
Di seguito vengono illustrati alcuni metodi di validazione dei dati.

- **Trasformazione della stringa con sequenze di escape**

Per combattere questo tipo di attacco, tutto quello che bisogna fare è di trasformare la stringa di caratteri forniti dall'utente sostituendo i caratteri di apice e doppio apice con le corrispondenti sequenze di escape: \' e \". Come abbiamo visto nelle precedenti Unità di apprendimento, sia Php che MySQL interpretano questi caratteri come parte della stringa e non più come delimitatori, rendendo vano l'attacco.
Per eseguire questa sostituzione si fa uso della funzione predefinita di Php **mysql_real_escape_string** che riceve una stringa e la restituisce sostituendo i caratteri apice, doppio apice, backslash e la stringa nulla con le rispettive sequenze di escape.

```
$user  = mysql_real_escape_string($_POST["utente"]);
$pwd   = mysql_real_escape_string($_POST["password"]);
```

- **Validazione dei caratteri inseriti**

Con questa tecnica ciascun carattere delle stringhe fornite dall'utente viene controllata per vedere se corrisponde a un carattere alfabetico (maiuscolo o minuscolo) oppure a una cifra numerica.

```
$user  = $_POST["utente"];
$pwd   = $_POST["password"];
$ammessi  = "abcdefghijklmnopqrstuvwxyz";
$ammessi .= "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
$ammessi .= "0123456789";

$userok = true;
for( $pos=0; $pos<strlen($user) && $userok; $pos++ )
{
   $car = substr($user, $pos, 1);
   if ( strpos($ammessi, $car) === false ) $userok = false;
}

$pwdok = true;
for( $pos=0; $pos<strlen($pwd) && $pwdok; $pos++ )
{
   $car = substr($pwd, $pos, 1);
   if ( strpos($ammessi, $car) === false ) $pwdok = false;
}
```

```
if ($userok and pwdok )
{
   accesso ai dati
}
else
   echo "Inserimento dati non corretto";
```

Il precedente codice utilizza alcune funzioni stringa predefinite del linguaggio Php:

- **strlen** restituisce la lunghezza di una stringa;
- **substr** estrae una sottostringa da una stringa. *substr(stringa, start, len)* restituisce *len* caratteri di *stringa* a partire dalla posizione *start*. Il primo carattere della stringa occupa la posizione 0; in assenza di *len* vengono restituiti tutti i caratteri di *stringa* a partire da *start*.
- **strpos** trova la posizione della prima occorrenza di una stringa all'interno di un'altra stringa e restituisce *false* in caso di esito negativo della ricerca; nell'esempio restituisce la posizione occupata da *$car* in *$ammessi*.

Il valore restituito da *strpos* deve essere trattato con cautela perché che il valore 0, in certi contesti, viene interpretato come il valore logico falso.

Questa situazione si presenta, per esempio, nel precedente codice quando si controlla che i caratteri di *$user* compaiano in *$ammessi*. Se in *$user* c'è il carattere 'a', la funzione *strpos*, ricercandolo in *$ammessi*, restituisce il valore 0 e il successivo confronto con *false* è vero, come se 'a' fosse un carattere non ammesso.

Per ovviare a questa situazione, nel confrontare il valore restituito da *strpos* con *false*, si è usato l'operatore **===** , che restituisce *true* se i valori confrontati concordano per valore e tipo. Quindi: (0 == *false)* è vera, ma (0 === *false*) è falsa; analogamente: (1 == 1.0) è vera, mentre (1 === 1.0) è falsa.

È interessante osservare che questo controllo previene anche un attacco nel quale l'utente, per forzare il sistema, digita come password sequenze del tipo: 1 or 1 = 1. Questo attacco potrebbe avere successo nel caso in cui, attendendo password di tipo numerico, si costruisce la clausola WHERE della SELECT che controlla utente e password come: "WHERE username = $user and password =$pwd" senza delimitare *$user* e *$pwd* con apici perché si tratta di valori numerici.

- **Validazione con controllo dei comandi SQL**

La terza modalità controlla che le stringhe fornite dall'utente non contengano comandi SQL oppure il carattere apice, oppure i caratteri – – che in molte versioni del linguaggio SQL indicano l'inizio di frasi di commento; l'inserimento di un commento, infatti, fa perdere il significato di codice alla rimanente parte del comando SQL previsto dal programmatore nell'applicazione.

Le parole da controllare sono contenute in un array di 8 elementi. Anche questo codice utilizza la funzione *strpos* per la ricerca di una sottostringa all'interno di una stringa.

```
$user  = $_POST["utente"];
$pwd   = $_POST["password"];

$vietate = array ("select", "insert", "update", "delete",
                  "drop",   "alter",  "–",      "");
```

```
$userok = true;
for( $k=0; $k <= 7 && $userok; $k++ )
  if ( strpos($vietate[$k], $user) !== false ) $userok = false;

$pwdok = true;
for( $k=0; $k <= 7 && $pwdok; $k++ )
  if ( strpos($vietate[$k], $pwd) !== false ) $userok = false;
// ---- prosegue come nel caso precedente
```

Si noti l'uso dell'operatore di confronto != =, detto *non identico*, secondo il quale il confronto è vero se i due termini del confronto differiscono per valore o tipo. (*0 !== false*) è vero, mentre (*0 != false*) è falso.

• Buffer overflow

Un altro tipo di attacco è effettuato immettendo nelle caselle di testo di una maschera stringhe di caratteri molto lunghe, nella speranza che una stringa di lunghezza superiore a quella prevista nello script o nel database invada la memoria con effetti imprevedibili. La contromisura da adottare per questo tipo di attacco è di troncare la stringa al numero di caratteri previsto per un dato campo.

Per estrarre una sottostringa da una stringa si usa la funzione predefinita *substr*. Per esempio, se per utente e password sono previsti 15 caratteri, per prevenire un attacco di questo tipo bisogna ricordarsi di troncare le stringhe ricevute come valore di questi parametri ai primi 15 caratteri.

```
$user  = substr( $_POST["utente"], 0, 15);
$pwd   = substr( $_POST["password"], 0, 15);
```