



Applicazioni avanzate di SQL-Injection

Antonio Parata
collaboratore OWASP-Italy
<http://www.ictsc.it>
antonio.parata@ictsc.it

OWASP

SMAU e-accademy

7 Ottobre 2006

<http://www.owasp.org/index.php/Italy>

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the OWASP License.

The OWASP Foundation
<http://www.owasp.org>

Introduzione all'SQL Injection

L'SQL Injection è un particolare tipo di attacco il cui scopo è quello di indurre il database ad eseguire query SQL non autorizzate.

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'

SQL Injection e metodi di Inferenza

Con il termine *Inferenza* si intende una conclusione tratta da un insieme di fatti o circostanze.

In pratica ciò che viene fatto è eseguire alcuni test di verità sui parametri vulnerabili e in base al risultato dedurre il valore del parametro.

I test effettuati sono di tipo binario (vero/falso).

Per poter eseguire con successo l'attacco è dunque necessario saper distinguere il significato di vero da quello di falso.

SQL Injection e metodi di Inferenza

Supponiamo che nell'applicazione che stiamo testando, vi sia un parametro vulnerabile, ad esempio *id*.

Per prima cosa creiamo una query sintatticamente corretta che restituisca un valore falso, ad esempio:

id = aaa' and '1' = '2

Con questa query otterremo sicuramente un valore falso. In particolare il valore falso è rappresentato dal contenuto della pagina web ritornata.

SQL Injection e metodi di Inferenza

Ciò che andremo a fare è cercare di determinare il valore di ogni singolo carattere del parametro di cui vogliamo scoprire il valore.

In pratica, ci chiederemo:

“Il carattere 1 del parametro è uguale ad a?” => NO

“Il carattere 1 del parametro è uguale a b?” => NO

“Il carattere 1 del parametro è uguale a c?” => NO

“Il carattere 1 del parametro è uguale a d?” => SI

Conosciamo il valore del primo carattere, procediamo con il secondo

“Il carattere 2 del parametro è uguale ad a?” => NO

“Il carattere 2 del parametro è uguale a b?” => SI

Conosciamo il valore del secondo carattere, procediamo in questo modo fino a scoprire i valori di tutti i caratteri che compongono il valore del parametro.

SQL Injection e metodi di Inferenza

Come facciamo a capire quando abbiamo esaminato tutta la stringa?

La funzione *substring* ritorna un valore null (pari al valore ASCII 0) quando l'indice da cui cominciare a considerare la sottostringa è maggiore della lunghezza della stringa stessa.

A questo punto basta verificare che il valore del carattere su cui stiamo facendo inferenza sia uguale a 0 per dedurre che abbiamo esaminato tutta la stringa.

Ma...

e se il valore della variabile su cui stiamo facendo inferenza contenesse il carattere ASCII 0?

Generalmente ciò non accade con gli input inseriti dall'utente, ma invece può accadere nel caso volessimo ricavare il valore di un file (binario).

SQL Injection e metodi di Inferenza

Soluzione: fare inferenza sulla lunghezza.

In pratica quando incontriamo il carattere *null*, andremo a fare inferenza sulla lunghezza della stringa, è inoltre necessario tenere memoria di quanti caratteri ho fino ad ora esaminato.

Le fasi da intraprendere sono:

Detto n il numero di caratteri fino ad ora analizzati, mi chiedo:

La lunghezza della stringa rappresentante il valore è uguale a n ?

SI => ho finito di fare inferenza

NO => il valore contiene il carattere ASCII 0, continuo a fare inferenza sul prossimo carattere.

SQL Injection e metodi di Inferenza

Un breve esempio.

*Goal: sappiamo che il parametro **id** è vulnerabile all'SQL Injection. Supponendo che il server sia eseguito con i privilegi di amministratore (root), sfruttiamo questa vulnerabilità per riuscire a ricavare il contenuto del file **/etc/shadow**. Inoltre supponiamo che il DBMS sia MySQL.*

A questo punto sappiamo come operare. La richieste che andremo ad eseguire sono della seguente forma:

```
http://www.mysite.com/index.php?id=20'%20
and%20ASCII(SUBSTRING(Load_File("/etc/shadow"),1,1))=0/*
```

Supponendo di aver ricavato 8 caratteri, e di aver incontrato il valore **null**, la query da eseguire per fare inferenza sul valore della lunghezza è:

```
http://www.mysite.com/index.php?id=20'%20
and%20CHAR_LENGTH(login)=8/*
```

Il tool: Sqlmap

Al momento supporta solo database Mysql (versione 5)

Permette di ottenere in automatico il valore di intere tabelle tramite inferenza

```
./sqlmap.py -u http://localhost/blind/showcontent.php?id=TOKEN -t TOKEN  
-d 1 -database blind -table news -dump -v
```

```
[*] starting at: 02:37:53
```

```
[02:37:53] fetching total attributes for db:blind table:news
```

```
[02:37:53] retrieved: 3
```

```
...
```

```
DATABASE: blind TABLE: news
```

```
— -  
|news                |id |highlight  
— -  
|ciao da daniele     |1  |numero 1  
|sempre ciao da daniele |2  |numero 2  
— -
```

```
[*] shutting down at: 02:37:58
```

Rendersi immuni all'SQL Injection

Nelle applicazioni web esistono due tipologie di parametri:

Parametri il cui valore deve essere fornito dall'utente (si pensi ad esempio alle form di autenticazione).

Parametri il cui valore è fisso e indipendente dalla sessione di navigazione (ad esempio i parametri che identificano l'oggetto nei vari negozi on-line).

Potremmo chiamare i primi **parametri dinamici** e i secondi **parametri statici**.

Per quanto riguarda i parametri dinamici, la loro validazione deve essere fatta ad-hoc dal progettista (ad esempio implementando una funzione di validazione basata su whitelist).

Concentriamoci invece sui parametri statici.

Rendersi immuni all'SQL Injection

Ciò che si fa è inserire un ulteriore parametro il quale identifica univocamente i restanti parametri della richiesta (sia che sia di tipo GET che di tipo POST).

Vediamo come fare in PHP (versione 5).

Utilizzando la class **PEAR::Crypt_HMAC**, possiamo utilizzare la funzione HMAC per creare un *hash* dei valori dei parametri. Per creare l'hash è necessaria una chiave segreta.

A questo punto tutti i parametri delle varie richieste saranno associati al rispettivo HMAC. Ogni volta che viene invocata una richiesta è necessario controllare il relativo HMAC per verificare che i parametri non sono stati alterati. Se sono stati alterati segnalare l'errore.

In questo modo non è possibile per un attacker modificare i parametri senza che l'applicazione se ne accorga.

Rendersi immuni all'SQL Injection

```
<?php
require_once('Crypt/HMAC.php');

Function create_parameters($array) {
    $clearText = "";
    $arrayParameters = array();

    /* Costruisco la stringa con la coppia chiave valore */
    foreach($array as $paramName => $paramValue) {
        $clearText = $clearText.$paramName.$paramValue;
        $arrayParameters[ ] = "$paramName=$paramValue";
    }

    $hmac = new Crypt_HMAC(SECRET_KEY, 'sha1');
    $hashValue = $hmac->hash($clearText);
    $arrayParameters[ ] = "hash=$hashValue";
    $queryString = join (&';', $arrayParameters);

    return $queryString;
}
```

Renderi immuni all'SQL Injection

```
<?php
require_once('Crypt/HMAC.php');

Function verify_parameters($array) {
    $clearText = "";
    $arrayParameters = array();

    if (!in_array('hash',$array) {
        return FALSE;
    }

    $hash = $array['hash'];
    unset($array['hash']);
    foreach($array as $paramName => $paramValue) {
        $clearText = $clearText.$paramName.$paramValue;
        $arrayParameters[ ] = "$paramName=$paramValue";
    }

    $hmac = new Crypt_HMAC(SECRET_KEY, 'sha1');
    if ($hash != $hmac->hash(SECRET_KEY) {
        return FALSE;
    }
    else {
        return TRUE;
    }
}
```

Rendersi immuni all'SQL Injection

Vediamo come creare un nuovo link:

```
<?php  
  
define('SECRET_KEY', "un6r34k4b13"); // Inserire la propria chiave segreta  
  
echo '<a href="script.php?".create_parameters(array('name' => 'value')).">link</a>  
?>
```

Per la verifica invece:

```
<?php  
  
define('SECRET_KEY', "un6r34k4b13"); // Inserire la propria chiave segreta  
  
if (!verify_parameters($arrayParametriRichiesta) {  
    die("VIOLAZIONE DELLA SICUREZZA!");  
}  
// altro...  
?>
```

Grazie per l'attenzione.