## 案例四：自动封IP和解IP

自09年踏入运维领域，到目前我所运维的服务器以及网站以PHP的居多，而前端以Nginx为主。在运维生涯里，出现最普遍的问题就是网站访问慢甚至访问不到，到服务器上查看后得出的结论是，这个网站被CC攻 击了。有的朋友可能不知道什么叫CC攻 击，在这里我简单介绍一下。CC攻 击属于DDos攻 击的一种，\*\*\*者会利用大量“肉鸡”对攻 击目标网站发起请求，并且频率很快，这样会导致目标网站的服务器承受不住而瘫痪。

CC攻击虽然看起来跟正常的访问几乎没有什么区别，但如果我们仔细分析访问日志还是可以找到一些线索，比如某个IP访问频次很高，或者某几个IP的user\_agent是固定的等等特性，有的甚至会去模仿正规的搜索引擎，比如，把自己伪装为百度的“蜘蛛爬虫”。当遇到CC攻 击时，只要你肯花费一些精力来分析访问日志，终究是可以找出发起CC攻 击的真凶，然后我们只要封掉它们的IP就万事大吉了。本案例的需求是这样的：

1）每分钟分析一次访问日志/data/logs/access\_log，日志片段如下：

180.98.113.151 - [19/Sep/2018:09:30:07 +0800] "/uc\_server/avatar.php?uid=1145811&amp;size=middle" 301 "GET HTTP/1.1" "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 9\_2\_1 like Mac OS X) AppleWebKit/601.1.46 (KHTML, like Gecko) Mobile/13D15 MAGAPPX|4.1.2-4.1.0-41|iPhone OS 9.2.1 iPhone 6|wenyou|C6C25422-279C-4337-8E10-F588D577B9D7|da97ede5be797f79b96d6761bf858632|426ef86c3fc2359dc90468f7bdd0f5e9|c64f2225ec641231cd612bbe08f2b40d"

61.227.224.229 - [19/Sep/2018:09:30:07 +0800] "/misc.php?mod=ranklist&amp;type=member&amp;view=post" 200 "GET HTTP/1.1" "http://www.wenyou.com/" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0"

183.207.95.145 [19/Sep/2018:09:30:07 +0800] "/uc\_server/avatar.php?uid=1323875&amp;size=middle" 301 "GET HTTP/1.1" "http://app.yikaidai.com/mag/circle/v1/forum/threadViewPage?tid=3446714&amp;circle\_id=&amp;themecolor=1aadfa" "Mozilla/5.0 (iPhone; CPU iPhone OS 11\_0\_3 like Mac OS X) AppleWebKit/604.1.38 "

114.230.251.50 - [19/Sep/2018:09:30:07 +0800] "/core/attachment/attachment/img?url=https%3A%2F%2Fmmbiz.qpic.cn%2Fmmbiz\_jpg" 302 "GET HTTP/1.1" "https://app.yikai.com/mag/info/v1/info/infoView?id=55855&amp;themecolor=1aadfa" "Mozilla/5.0 (iPhone; CPU iPhone OS 11\_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) "

61.227.224.229 - [19/Sep/2018:09:30:07 +0800] "/misc.php?mod=ranklist&amp;type=member&amp;view=onlinetime" 200 "GET HTTP/1.1" "http://www.wenyou.com/" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0"

2）把访问量高于100的IP给封掉

3）封过的IP都要记录到一个日志中

4）每隔30分钟检查一次被封的IP，把没有访问量或者访问量很少的IP解封

5）解封的IP记录到另外一个日志中

知识点一：awk用法

在shell脚本中，awk出现概率是极高的，因为它在处理字符串上有很强的能力。先来看一个小例子：

# awk -F ':' '$3>500 {print $1,$3}' /etc/passwd

systemd-bus-proxy 999

polkitd 998

chrony 997

user1 1000

说明：awk最核心的功能是分段，可以用-F选项指定一个分隔符，然后针对某一段字符进行处理，本例中用':'作为分隔符，去找第3段大于500的行，然后把第1段和第3段打印出来。在awk中可以使用>, &lt;, >=, &lt;=, ==, !=等逻辑判断的符号，这和shell是有差异的。本例中>后面的500一定不要加双引号，否则它就不以数字作为比较对象了，而是把500当成是字符串，结果自然就不一样了。如下：

# awk -F ':' '$3>"500" {print $1,$3}' /etc/passwd

shutdown 6

halt 7

mail 8

nobody 99

systemd-bus-proxy 999

dbus 81

polkitd 998

tss 59

postfix 89

sshd 74

chrony 997

awk的功能不止于此，它实际上跟shell一样属于一门脚本语言，可以写脚本。它的作者设计它的初衷是为了去格式化输出文本，它可以满足各种复杂的格式需求，不过我们平时写shell脚本时，仅仅把它作为一个命令来处理字符串，下面我列几个常见的用法（以下所有演示文件都使用1.txt）。

1）截取指定段

# awk -F ':|#' '{print $2}' 1.txt

说明：分隔符可以是一个正则表达式，本例中的分隔符可以是':'也可以是'#'。

2）匹配字符后字符串

# awk -F ':' '$1 ~ "abc"' 1.txt

说明：过滤出第一段包含abc的行，其中这里的abc可以是一个正则表达式，例如：

# awk -F ':' '$1 ~ "^ro+"' 1.txt

说明：^表示开头，+表示+前面的字符至少有1个，所以"^ro+"可以匹配的字符串有：ro, roo, rooo...

3）多个语句同时使用

# awk -F ':' '$1 ~ "root" {print $1,$3}; $3>100 {print $1,$2}' 1.txt

4）多个条件</p>

# awk '$3&lt;100 &amp;&amp; $7 ~ "bash" {print $0}' 1.txt

说明：如果不指定分隔符，则以空白字符作为分割符，在awk中可以用”;”表示并且，用||表示或者。$0会输出整行。

5）内置变量

# awk -F ':' 'print NF,NR' 1.txt

说明：NF为段数，NR为行数

6）数学运算

# awk -F ':' '$7=$3+$4 {print $0}' 1.txt

因为$7的值被重新定义了，所以打印$0时，并不会包含分隔符，当然也可以定义分隔符：

# awk -F ':' -v OFS='#' '$7=$3+$4 {print $0}' 1.txt

说明：用-v定义变量，这个OFS也是一个内置变量，它表示输出的结果的分隔符。

计算某一段的总和，这种用法也比较常见，例如：

# awk -F ':' '{(tot=tot+$3)}; END {print tot}' 1.txt

知识点二：sort排序

语法： sort [-t 分隔符] [-kn1,n2] [-nru] 这里的n1 &lt; n2

-t 分隔符 ：作用跟cut的-d一个意思<br/>-n ：使用纯数字排序

-r ：反向排序

-u ：去重复

-kn1,n2 ：由n1区间排序到n2区间，可以只写-kn1，即对n1字段排序

如果sort不加任何选项，则从首字符向后，依次按ASCII码值进行比较，最后将他们按升序输出，如下：

# head -n5 /etc/passwd |sort

adm:x:3:4:adm:/var/adm:/sbin/nologin

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

root:x:0:0:root:/root:/bin/bash

-t 后面跟分隔符，-k后面跟数字，表示对第几个区域的字符串排序，-n 则表示使用纯数字排序，示例如下：

# head -n5 /etc/passwd |sort -t: -k3 -n

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

-k3,5 表示从第3到第5区域间的字符串排序，-r表示反向排序，示例如下：

# head -n5 /etc/passwd |sort -t: -k3,5 -r

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

bin:x:1:1:bin:/bin:/sbin/nologin

root:x:0:0:root:/root:/bin/bash

知识点三：uniq去重复

这个命令我经常用的选项只有一个，-c ：统计重复的行数，并把行数写在前面

请把下面的内容写入testb.txt, 保存。

111

222

111

333

使用uniq 的前提是需要先给文件排序，否则不管用，如下所示：

# uniq testb.txt

111

222

111

333

# sort testb.txt |uniq

111

222

333

以下是-c选项的作用：

# sort testb.txt |uniq -c

2 111

1 222

1 333

知识点四：用iptables工具封/解封IP

在CentOS系统中有一个叫做netfilter的防火墙，它可以对进入或者即将离开网卡的数据包进行处理。最常见的用法就封IP，即限制某些IP或者某些IP的某个port的数据包进入Linux系统。很多人把iptables叫做防火墙，其实不然，iptables是CentOS6系统里的一个服务，当然它也是一个命令，而在CentOS7系统里也有iptables命令，服务名不再是iptables，而是firewalld。

1）查看filter表的iptables规则

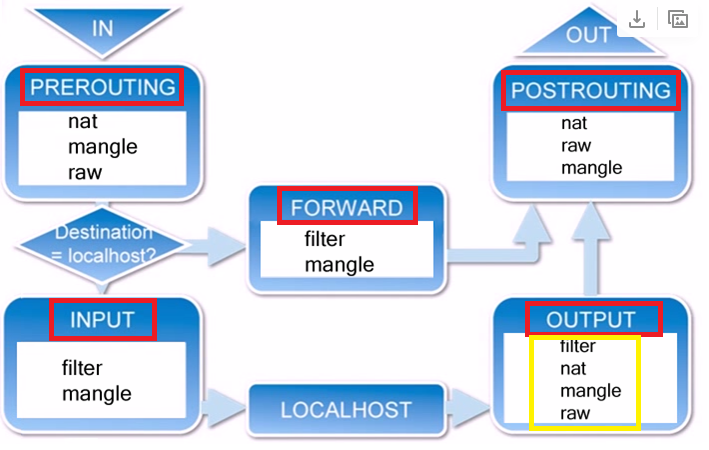
# iptables -nvL -t filter

说明：在CentOS7上netfilter有5个表，分别是：filter、nat、mangle、raw、secuirty。而我们用的比较多的是前两个，本案例中的封ip用的就是filter表。该命令中的-t filter可以省略，默认就是filter表。

2）增加规则

# iptables -A INPUT -p tcp --dport 80 -s 1.1.1.1 -j DROP

说明： -A表示增加规则，INPUT为filter表里的一个链，除此之外，filter表还有OUTPUT链和FORWARD链。这里我引用网上的一张图片帮助大家理解netfilter的表和链。



数据包的流向是这样的：

a. 当一个数据进入网卡时，他首先进入PREOUTING链（数据包进入路由之前），然后判断目标IP是否本机。

b. 如果数据包是进入本机的，他会到达INPUT链（通过路由表后目的地为本机），数据包到达INPUT链后进入本机内核，然后内核进行处理，处理完到OUTPUT链（由本机产生，向外转变），最后到POSTROUTING（发送到网卡接口之前）。

c. 如果不是进入本机的，他会到FORWARDING链（通过路由表后，目的地不为本机）最后POSTROUTING链（发送到网卡接口之前）。

搞清楚了数据包流向就很容易理解INPUT、OUTPUT链的作用了，其实我们更关心的是INPUT和OUTPUT链，而其他几个链可以先不用理会。用INPUT控制进入本机的数据包，而用OUTPUT控制出去的数据包，这两个几乎可以满足我们绝大多数的运维场景了。

-A选项是增加一条规则，更准确地说是追加一条规则，因为iptables的规则是分前后的，用iptables -nvL查看规则时，越靠上的规则就越先生效，也就是说优先级越高。所以，使用-A追加的规则只能排在最后面，但如果想增加一条优先级最高的规则怎么办？用-I选项（插入规则的意思）即可，示例命令如下：

# iptables -I INPUT -p tcp --dport 80 -s 2.2.2.2 -j DROP

也可以不针对端口，只写IP，同时也可以指定目标IP地址，如下：

# iptables -I INPUT -s 3.3.3.3 -d 4.4.4.4 -j DROP

删除规则有两个方法，一是全部清空，另外一个就是删除一条：

# iptables -t filter -F

删除一条规则，比较麻烦，因为你需要记得当时创建这条规则的命令，把-I或者-A换成-D即可，如下：

# iptables -D INPUT -s 3.3.3.3 -d 4.4.4.4 -j DROP

当然，也有比较简单的做法，首先要列出规则的序号，如下：

# iptables -nvL --line-numbers

最左边第一列的&quot;num&quot;就是规则的序号了，每个链的规则都是从1开始，假如要删除INPUT链的第2条规则，只需要执行：

# iptables -D INPUT 2

这样就会把INPUT链下的第2条规则删除掉。删除规则后，重新列出规则序号，会发现序号依然是从1开始。

当我们执行iptables -nvL时，列出的结果中第一列'pkts'表示被iptables规则作用的数据包个数，而第二列'bytes'表示被iptables规则作用的数据量大小。这个数字会不断变化，如果重启iptables或者firewalld服务这两列数字会清零，当然也有一个选项可以让它清零，如下：

# iptables -Z

也可以指定某个链，比如：

# iptables -Z INPUT

以上我介绍的知识点主要是针对本案例的，其实iptables的用法还挺多，限于篇幅暂时先讲这么多吧，后续案例中有涉及到咱们再近一步学习。

知识点五：shell脚本中的函数

在shell脚本中如果某一段代码出现1次以上，应该把这段代码封装到一个函数里，这样后续调用它会很方便，而且代码看起来也非常美观。下面我用一个实际的例子来说明函数的好处。

需求是，检查系统中是否安装vim-enhanced、expect、wget包，如果没有安装则安装一下，传统的shell代码是这样写的：

if ! rpm -q vim-enhanced &>;/dev/null

then

yum install -y vim-enhanced

fi

if ! rpm -q expect &>;/dev/null

then

yum install -y expect

fi

if ! rpm -q wget &>;/dev/null

then

yum install -y wget

fi

试问，如果让你去检查100个包，难道你要写100遍这段代码吗？用函数就简洁多了，如下：

## 定义if\_install函数

if\_install() {

if ! rpm -q $1 &>;/dev/null

then

yum install -y $1

fi

}

## 用for循环把所有rpm包逐一检查一遍

for pkg in vim-enhanced expect wget

do

if\_install $pkg

done

知识点六：sed用法

sed和grep、awk一样频繁地出现在shell脚本中，它们三个经常被叫做“正则三剑客”，可见这三个工具和正则表达式之间的关系非同一般。对于sed工具的用法，我列几个常见的。

1）打印指定行

sed -n '10p' 1.txt

说明：10指的是行号，p表示print（打印），加上-n后只显示第10行，否则会把1.txt所有行都显示出来，这里的单引号是我的习惯，你可以不加也可以加双引号。另外，可以指定多行，示例命令如下：

sed -n '1,5p' 1.txt

说明：打印1到5行，如果是不连续的行，可以这样：

sed -n '1p;5p' 1.txt

sed也有和grep类似的过滤指定行的功能，如下：

sed -n '/abc/p' 1.txt

说明：//为一个固定写法，里面就是要匹配的字符串，可以是正则，例如：

sed -n '/[a-z0-9]/p' 1.txt

在正则表达式中，有几个特殊符号属于扩展正则范畴，它们是+, ?, |, (), {}。在grep中要使用它们需要加上-E选项，而在sed中要使用它们，需要加上-r选项，比如：

sed -nr '/abc|123/p' 1.txt

2）删除指定行

sed '10d' 1.txt

说明：会删除第10行，但并不会真正操作1.txt内容，而是屏幕上显示的内容少了第10行，要想直接在文件内生效，可以加上-i选项，如下：

sed -i '10d' 1.txt

删除包含'abc'或者'linux'字符串的行

sed -ir '/abc|linux/d' 1.txt

3）查找替换

把1.txt中的出现的全部'aming'替换为'linux'

sed 's/aming/linux/g' 1.txt

说明：这里的s表示替换，g表示全局替换，如果不加g则只替换每行中出现的第一个'aming'。

本案例参考脚本

如下是我写的参考脚本，请过目：

#! /bin/bash

## 把访问量比较大的IP封掉，如果20分钟内被封的IP没有请求或者请求很少，需要解封

## 作者：

## 日期：2018-09-20

## 版本：v0.1

#定义1分钟以前的时间，用于过滤1分钟以前的日志

t1=`date -d "-1 min" +%Y:%H:%M`

log=/data/logs/access\_log

block\_ip()

{

egrep "$t1:[0-5]+" $log > /tmp/tmp\_last\_min.log

#把1分钟内访问量高于100的ip记录到一个临时文件中

awk '{print $1}' /tmp/tmp\_last\_min.log |sort -n |uniq -c|sort -n |awk '$1>100 {print $2}' > /tmp/bad\_ip.list

#计算ip的数量

n=`wc -l /tmp/bad\_ip.list|awk '{print $1}'`

#当ip数大于0时，才会用iptables封掉它

if [ $n -ne 0 ]

then

for ip in `cat /tmp/bad\_ip.list`

do

iptables -I INPUT -s $ip -j REJECT

done

#将这些被封的IP记录到日志里

echo "`date` 封掉的IP有：" >> /tmp/block\_ip.log

cat /tmp/bad\_ip.list >> /tmp/block\_ip.log

fi

}

unblock\_ip()

{

#首先将包个数小于5的ip记录到一个临时文件里，把它们标记为白名单IP

iptables -nvL INPUT|sed '1d' |awk '$1&lt;5 {print $8}' > /tmp/good\_ip.list

n=`wc -l /tmp/good\_ip.list|awk '{print $1}'`

if [ $n -ne 0 ]

then

for ip in `cat /tmp/good\_ip.list`

do

iptables -D INPUT -s $ip -j REJECT

done

echo "`date` 解封的IP有：" >> /tmp/unblock\_ip.log

cat /tmp/good\_ip.list >> /tmp/unblock\_ip.log

fi

#当解封完白名单IP后，将计数器清零，进入下一个计数周期

iptables -Z

}

#取当前时间的分钟数

t=`date +%M`

#当分钟数为00或者30时（即每隔30分钟），执行解封IP的函数，其他时间只执行封IP的函数