

01.Iptables防火墙实践

01.Iptables防火墙实践

1.防火墙基本概述

1.1.什么是防火墙

1.2.防火墙种类

2.Iptables基本介绍

2.1.什么是iptables

2.2.什么是包过滤防火墙

2.3.包过滤防火墙如何实现

3.Iptables链的概念

3.1.什么是链

3.2.iptables有哪些链

4.Iptables表的概念

4.1.什么是表

4.2.表的功能

4.3.表与链的关系

4.4.表与链相关问题

5.Iptables规则管理

5.1.什么是规则

5.2.如何查看规则

5.3.如何添加规则

5.4.如何修改规则

5.5.如何删除规则

5.6.如何保存规则

6.Iptables基本匹配

6.1.iptables匹配参数

6.2 iptables匹配场景

7.Iptables扩展匹配

7.1.multiport模块

7.2.iprange模块

7.3.string模块

7.4.time模块

7.5.icmp模块

7.7.connlimit模块

7.6.limit模块

7.6.tcp-flags模块

7.8.state模块

8.Iptables地址转换

8.1.什么是NAT

- 8.2 NAT的几种模式
- 8.3 NAT环境搭建
- 8.3.SNAT场景示例
- 8.4.DNAT场景示例
- 9.Iptables自定义链
 - 9.1.为什么要使用自定义链
 - 9.2.自定义链基本应用
 - 9.3.自定义链执行顺序
 - 9.4.如何删除自定义链

02.SSH远程管理服务实战

- 1.SSH基本概述
 - 1.什么是SSH
 - 2.SSH服务主要提供什么功能
 - 3.SSH远程连接 VS Telnet远程连接的区别?
- 2.SSH客户端命令
 - 2.1 ssh远程登陆
 - 2.2 scp远程拷贝
- 3.SSH验证方式
 - 3.1 基于密码验证
 - 3.2 基于秘钥验证
 - 3.2.1 创建密钥
 - 3.2.2 推送公钥
 - 3.2.3 测试连接
- 4.SSH模拟跳板机
 - 4.1 Windows下发密钥
 - 4.2 跳板机下发密钥
- 5.SSH安全优化
- 6.SSH免密Web

1.防火墙基本概述

1.1.什么是防火墙

- 过去，很长一段时期里，房屋都是砖木结构甚至是茅屋。如果一家失火，四邻也会跟着遭殃，所以为安全起见，古人就在自己居住地周围修筑高高的围墙，以阻挡外来的火势，保护自身的安全，这种墙就叫“防火墙”。
- 如今，“因特网”把世界各地的计算机都紧密地连接在一起。如果我们不严加防卫，一旦网络侵害，可能会出现不可预计的损失。那么在互联网上，我们会采用类似防火墙的方法，来保护我们的网络不受外来攻击或者侵害、为此我们需要设定防火墙规则，确定哪些类型的数据允许通过，哪些不允许通过。那么具有这种功能的设备或软件就称为“防火墙”。

1.2. 防火墙种类

- 从逻辑上讲，防火墙可以大体分为主机防火墙和网络防火墙。
 - 主机防火墙：针对于单个主机进行防护，比如Windows。
 - 网络防火墙：往往处于网络入口或边缘，针对于网络入口进行防护，服务于防火墙背后的服务器集群。
- 从物理上讲，防火墙可以分为硬件防火墙和软件防火墙。
 - 硬件防火墙：在硬件级别实现部分防火墙功能，另一部分功能基于软件实现，性能高，成本高。
 - 软件防火墙：以软件的方式模拟防火墙功能，运行在操作系统上，性能不高，成本较低。

2.Iptables基本介绍

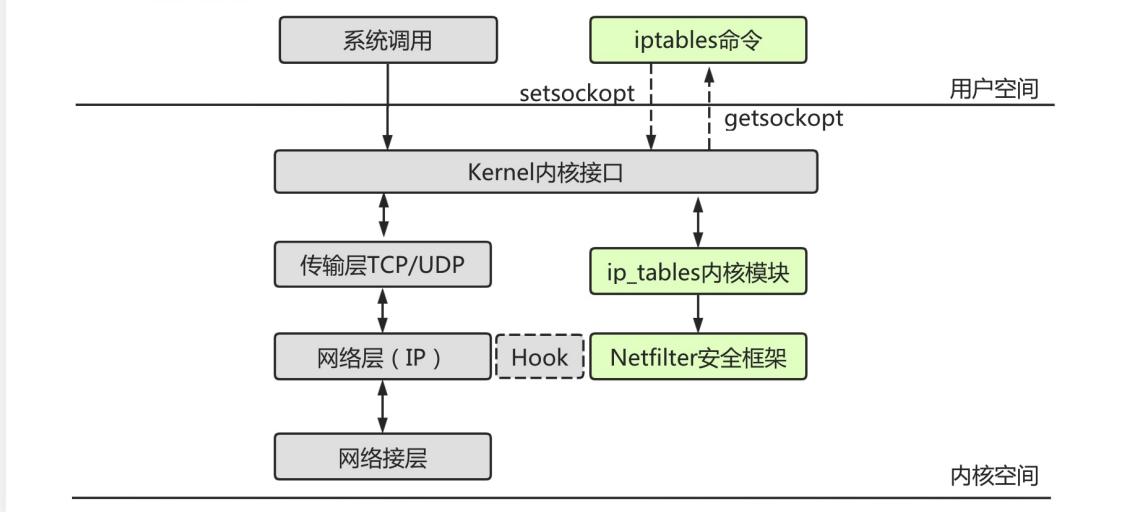
2.1. 什么是iptables

`iptables` 其实不是真正的防火墙，我们可以把它理解成是一个代理程序，用户通过 `iptables` 这个代理程序，将安全规则执行到对应的“安全框架”中，这个“安全框架”才是真正的防火墙；这个安全框架叫 `netfilter`

`netfilter` 位于操作系统的内核空间。`iptables` 位于操作系统的用户空间，我们后期是通过 `iptables` 命令工具操作 `netfilter` 内核框架。

Netfilter/Iptables关系

--oldxu



`netfilter/iptables` 是 `linux` 平台下的“包过滤型防火墙”，这个包过滤防火墙是免费的，它可以代替昂贵的商业防火墙解决方案，完成数据包的过滤、网络地址转换（NAT）等功能。

2.2.什么是包过滤防火墙

- 包过滤防火墙它在网络层截获网络数据包的包头（header），它针对数据包的包头，根据预先定义好的防火墙过滤规则进行比对，与规则相匹配的包会被放行，与规则不匹配的包则可能会被丢弃、也可能执行更复杂的动作。
- 包过滤防火墙一般作用在网络层，故也称“网络层防火墙”，通过检查数据流中每一个数据包的源IP地址、目的IP地址、源端口、目的端口、协议类型（TCP、UDP、ICMP）、状态等信息来判断是否符合规则。

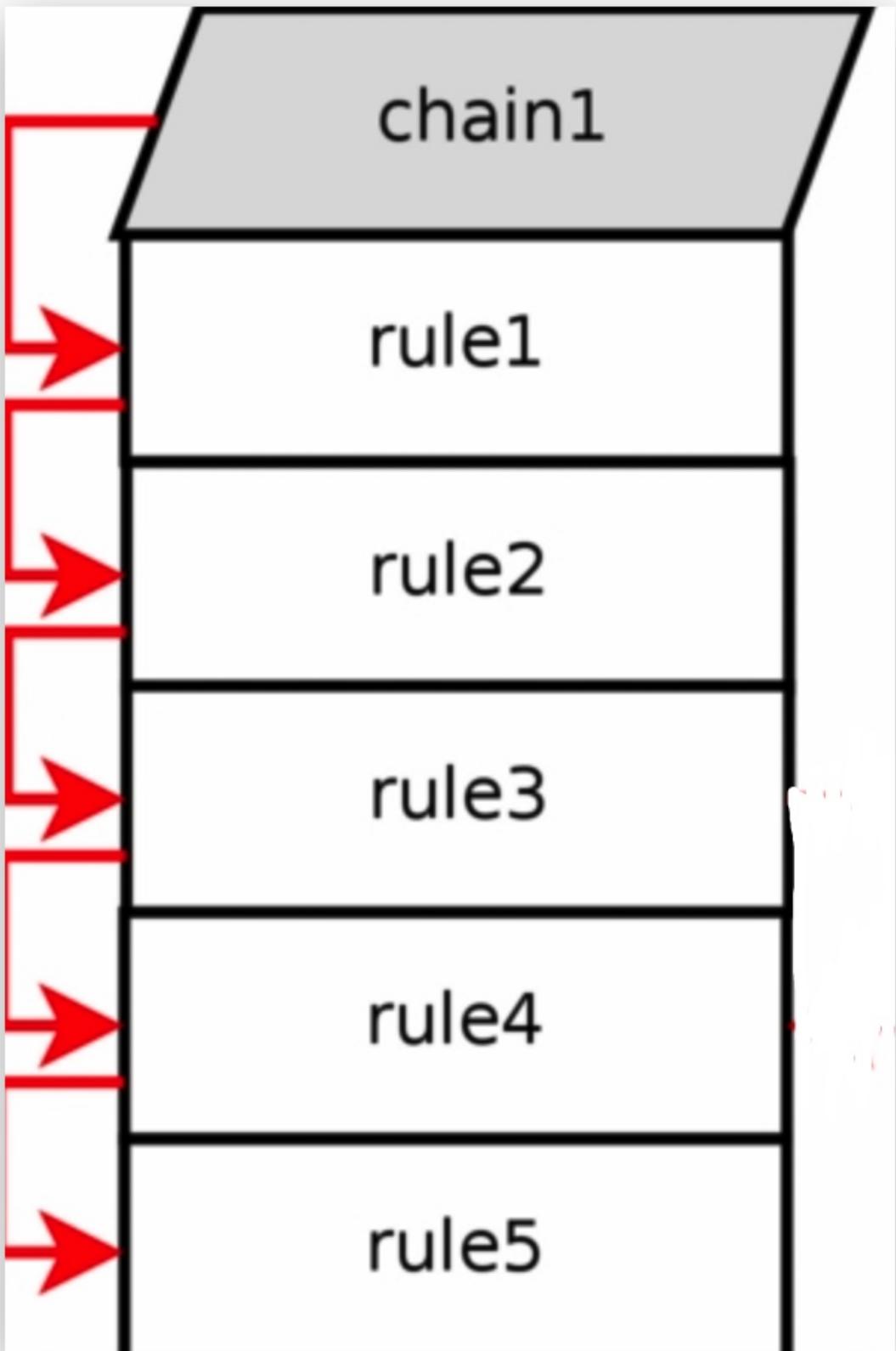
2.3.包过滤防火墙如何实现

- 包过滤防火墙是由 Netfilter 来实现的，它是内核的一部分，所以，如果我们想要防火墙能够达到“防火”的目的，则需要在内核中设置关卡，所有进出的报文都要经过这些关卡，进行检查，将符合条件的放行，不符合条件的阻止，而这些关卡在iptables中不被称为“关卡”，而被称为“链”。

3.Iptables链的概念

3.1.什么是链

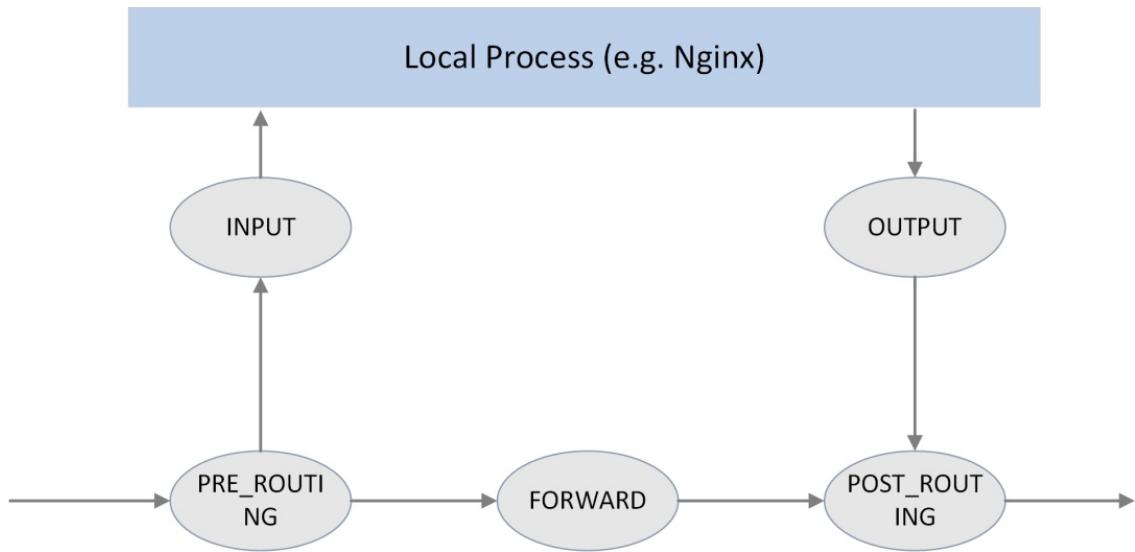
- 在 iptables 中的关卡为什么被称作“链”呢？
- 我们知道，防火墙的作用就在于对经过的数据报文进行“规则”匹配，然后执行规则对应的“动作”，所以当报文经过这些关卡的时候，则必须匹配这个关卡上的规则，但是，这个关卡上可能不止有一条规则，而是有很多条规则，当我们把这些规则串到一起的时候，就形成了“链”
- 所以，每个经过这个“关卡”的报文，都要将这条“链”上的所有规则匹配一遍，如果有符合条件的规则，则执行规则对应的动作，如果没有则执行默认链的动作。



3.2.iptables有哪些链

- 当我们启用了防火墙功能时，报文需要经过很多关卡，也就是说，根据实际情况的不同，报文经过"链"可能不同。
 - 请求本机会经过哪些链 (PREROUTING-->INPUT-->Local Process) ;
 - 经过本机又会经过哪些链 (PREROUTING-->FORWARD-->POSTROUTING) ;

- 从本机发出又会经过哪些链 (Local Process-->OUTPUT-->POSTROUTING) ;
- 了解Iptables链的数据包流向；后期在设定规则的时候，就知道往哪个链上面添加规则；



- 通过上图我们总结了 `iptables` 总共有五个链，分别是 `PREROUTING`、`INPUT`、`OUTPUT`、`FORWARD`、`POSTROUTING`。

4.Iptables表的概念

4.1.什么是表

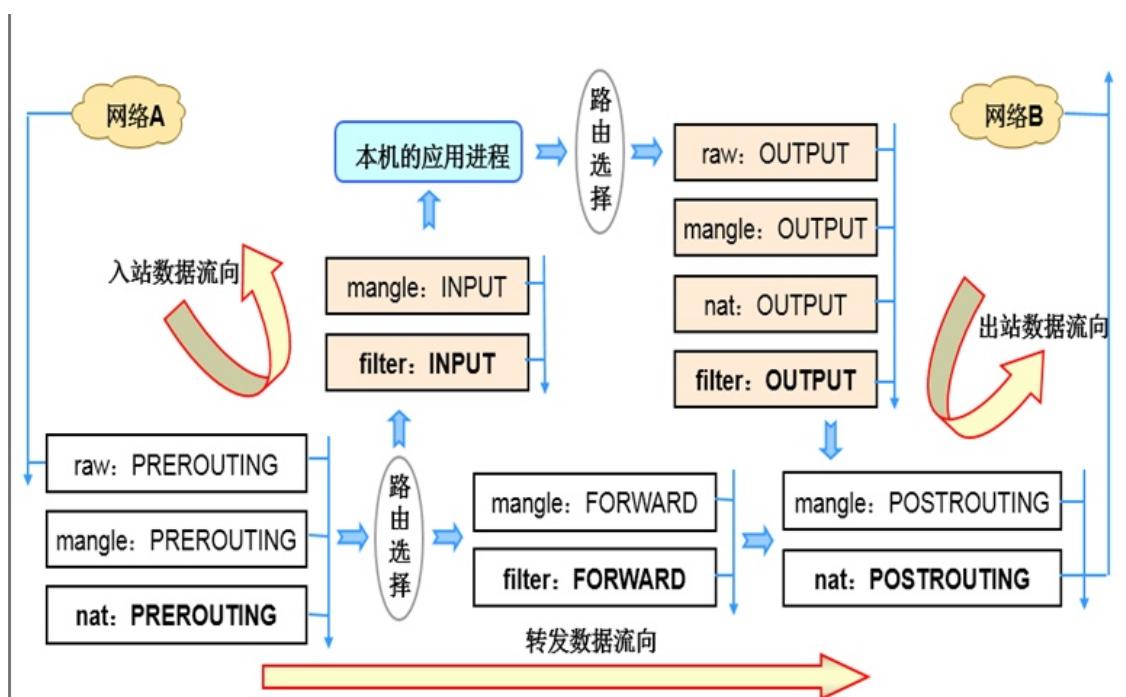
- 我们对每个"链"上都放置了一串规则，但是这些规则有些很相似，比如，A类规则都是对IP或者端口的过滤，B类规则是修改报文，那么这个时候，我们是不是能把实现相同功能的规则放在一起呢？
- 我们把具有相同功能规则的集合叫做"表"，所以说，不同功能的规则，我们可以放置在不同的表中进行管理，而 `iptables` 已经为我们定义了4种表，每种表对应了不同的功能。

4.2.表的功能

表名	作用	包含的链
filter	负责过滤功能	INPUT、OUTPUT、FORWARD
nat	负责网络地址转换功能	PREROUTING、INPUT、OUTPUT、POSTROUTING
mangle	负责修改数据包内容	INPUT, OUTPUT、FORWARD、POSTROUTING、PREROUTING
raw	关闭nat表上启用的连接追踪机制	PREROUTING、OUTPUT

4.3. 表与链的关系

- raw --> mangle --> nat --> filter



4.4. 表与链相关问题

- 问题1：来自于 10.0.0.1 的地址，访问本机的 web 服务请求都不允许，应该在哪个表的哪个链上设定规则？
- 答案1：很多同学会觉得是 PREROUTING 链，但其实是 INPUT 链。因为我们要做的是过滤，而 PREROUTING 不能做过滤，所以是 INPUT。
- 问题2：所有由本机发往 10.0.0.0/24 网段的 TCP 服务都不允许？
- 答案2：由本地发出会经过 OUTPUT、POSTROUTING，但由于 POSTROUTING 不支持做过滤，所以应该在 OUTPUT 规则链上配置。
- 问题3：所有来自自己本地内部网络的主机，像互联网发送 web 服务器请求都允许？

- 答案3：本地内部主机发送互联网经过 PREROUTING、FORWARD、POSTROUTING 而能做过滤的只有 FORWARD。

5.Iptables规则管理

5.1.什么是规则

- 数据包的过滤基于规则。而规则由匹配条件+动作组成。
- 那么我们对规则的操作无非就是增、删、查、改。
- `iptables [-t表名] 选项 [链名] [规则] [动作]`,操作规则时的考量点：
 - 1) 要实现什么功能：判断添加到哪个表上；
 - 2) 报文流经的路线：判断添加到哪个链上；

iptables选项	含义	示例
-t, --table	指定要操作的表 (默认filter)	<code>iptables -t filter</code>
-A, --append	追加一条规则至链的末尾	<code>iptables -t filter -A INPUT</code>
-I, --insert	插入一条规则至链的顶部	<code>iptables -t filter -I INPUT</code>
-D, --delete	指定删除一条规则	<code>iptables -t filter -D INPUT 1</code>
-R, --replace	替换选定链中的规则	<code>iptables -t filter -R INPUT</code>
-S, --list-rules	打印选定链中的所有规则	<code>iptables -t filter -S</code>
-F, --flush	清空链中的所有规则	<code>iptables -t filter -F</code>
-Z, --zero	将所有链中的数据包和字节计数器归零	<code>iptables -t filter -Z</code>
-N, --new-chain	创建自定义名称规则链	<code>iptables -N New_Rules</code>
-E, --rename-chain	给自定义链修改名称	<code>iptables -E Old_Rules New_Rules</code>
-X, --delete-chain	删除自定义链	<code>iptables -X Rules_Name</code>
-P, --policy	给链设定默认策略	<code>iptables -t filter -P DROP</code>

5.2.如何查看规则

```
# -t指定表明、-L查看详情、-n不反解、-v详细信息、--line-numbers显示规则编号
[root@Route ~]# iptables -t filter -L -n -v --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
Chain FORWARD (policy ACCEPT)
num  target     prot opt source          destination
Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
```

5.3.如何添加规则

- 示例：允许其他任何主机 ping 通本机。

```
[root@Route ~]# iptables -t filter -I INPUT -p icmp -j ACCEPT
[root@Route ~]# iptables -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    ACCEPT     icmp  --  0.0.0.0/0      0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
```

5.4.如何修改规则

- 示例：修改规则，拒绝其他任何主机 ping 通本机。

```
[root@Route ~]# iptables -t filter -R INPUT 1 -p icmp -j DROP
[root@Route ~]# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0           0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

5.5.如何删除规则

- 方法一：根据规则的编号去删除规则
- 方法二：根据具体的匹配条件与动作删除规则

```
[root@Route ~]# iptables -t filter -D INPUT 1
[root@Route ~]# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0           0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

5.6.如何保存规则

1.默认 `iptables` 属于临时生效，所以需要将编写的规则永久存储起来；

```
[root@lb01 ~]# iptables-save >/etc/iptables.rules
```

2.清空 `iptables` 规则，然后恢复规则测试；

```
[root@lb01 ~]# iptables -F
[root@lb01 ~]# iptables-restore < /etc/iptables.rules
```

6.Iptables基本匹配

- 我们前面在练习规则操作时，使用的"匹配条件"比较少。下面我们来了解下 `iptables` 匹配条件更多的用法。

6.1.iptables匹配参数

条件参数	含义
[!] -p, --protocol protocol	指明需要匹配的协议，如icmp、udp、tcp
[!] -s, --source address[/mask] [,...]	指定匹配源地址，如有多个可以逗号分隔
[!] -d, --destination address[/mask][,...]	指定匹配目标地址，如有多个可以逗号分隔
[!] --source-port,--sport port[:port]	指定源端口
[!] --destination-port,--dport port[:port]	指定目标端口
[!] -i, --in-interface name	接收数据包的接口名称
[!] -o, --out-interface name	发送数据包的接口名称
-m, --match match	执行需要使用的匹配项，属于扩展匹配
-j, --jump target	执行匹配规则后的动作、ACCEPT、DROP、REJECT等

6.2 iptables匹配场景

- 示例1：仅允许 10.0.0.10 访问 10.0.0.200 服务器的 80 端口、其他地址全部拒绝。

```
# -I 插入规则至第一行、-A 追加规则、-s源地址、-d目标地址、--dport目标端口、-j匹配后执行的动作
[root@Route ~]# iptables -t filter -I INPUT -s 172.16.1.1 -d 172.16.1.61 -p tcp --dport 80 -j ACCEPT
[root@Route ~]# iptables -t filter -A INPUT -d 172.16.1.61 -p tcp --dport 80 -j DROP

Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    ACCEPT     tcp  --  172.16.1.1      172.16.1.61
tcp dpt:80
2    DROP       tcp  --  0.0.0.0/0      172.16.1.61
tcp dpt:80
```

- 示例2：所有来访问本机的协议，属于TCP协议的我们通通都放行；

```
[root@Route ~]# iptables -t filter -I INPUT -p tcp -j ACCEPT
[root@Route ~]# iptables -t filter -A INPUT -j DROP
```

- 示例3：凡是由本机发出的TCP协议报文，都允许出去，其他协议不行；

```
[root@Route ~]# iptables -I OUTOUT -p tcp -j ACCEPT
[root@Route ~]# iptables -A OUTOUT -j DROP
```

- 示例4：禁止其他主机从 eth0 向本机发送 ping 请求

```
[root@Route ~]# iptables -I INPUT -i eth0 -p icmp -j DROP
```

- 示例5：允许从本机发送 ping 请求，其他任何协议都不允许；

```
[root@Route ~]# iptables -I OUTPUT -p icmp -j ACCEPT
[root@Route ~]# iptables -I OUTOUT -j DROP
```

7.Iptables扩展匹配

- 前面我们已经对 `iptables` 中的基本匹配条件有了基本的认识，接下来，我们来认识一些新的扩展模块。

7.1.multiport模块

- 使用 `multiport` 模块可以添加多个不连续的端口; `-m multiport <-sports|--dports|--ports> 端口1[,端口2,...,端口n]`
- 示例：`10.0.0.10` 访问本机 20、21、80、443 允许通过；

```
[root@Route ~]# iptables -F
[root@route ~]# iptables -t filter -I INPUT -m multiport -s 10.0.0.10 -d 10.0.0.200 -p tcp --dports 20:22,80,443 -j ACCEPT

[root@route ~]# iptables -t filter -A INPUT -j DROP
Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    ACCEPT     tcp  --  10.0.0.10        0.0.0.0/0
multiport dports 20:22,80,443
2    DROP       tcp  --  0.0.0.0/0      0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source          destination
```

```
Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
```

7.2.iprange模块

- 使用 `iprange` 模块可以指定 "一段连续的IP地址范围"，用于匹配报文的源地址或者目标地址，`iprange` 扩展模块中有两个扩展匹配条件可以使用。
 - `[!] --src-range from[-to]`: 原地址范围
 - `[!] --dst-range from[-to]`: 目标地址范围
- 示例：`10.0.0.5-10.0.0.10` 地址段 ping 本机，则丢弃；

```
[root@route ~]# iptables -t filter -F
[root@route ~]# iptables -t filter -I INPUT -p icmp -m iprange --src-range "10.0.0.5-10.0.0.10" -j DROP
[root@route ~]# iptables -t filter -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    DROP       icmp  --  0.0.0.0/0           0.0.0.0/0
      source IP range 10.0.0.5-10.0.0.10

Chain FORWARD (policy ACCEPT)
num  target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
```

7.3.string模块

- 使用 `string` 扩展模块，可以指定要匹配的字符串，如果报文中包含对应的字符串，则符合匹配条件。
 - `--algo {bm|kmp}`: 字符匹配的查询算法；
 - `[!] --string pattern`: 字符匹配的字符串；
- 示例：应用返回的报文中包含字符 `"hello"`，我们就丢弃当前报文，其余正常通过。

```
# 安装httpd准备两个文件
[root@Route ~]# yum install httpd -y
[root@Route ~]# echo "hello" > /var/www/html/test.html
[root@Route ~]# echo "index-oldxu" > /var/www/html/index.html
[root@Route ~]# systemctl start httpd

# 配置规则
[root@route ~]# iptables -F
```

```
[root@route ~]# iptables -t filter -I OUTPUT -p tcp -m string --string "hello" --algo kmp -j DROP
[root@route ~]# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
DROP      tcp   --  0.0.0.0/0           0.0.0.0/0
STRING match  "hello" ALGO name kmp TO 65535
```

7.4.time模块

- 使用time扩展模块，根据时间段匹配报文，如果报文到达的时间在指定的时间范围以内，则符合匹配条件。
 - `--timestart hh:mm[:ss]`: 开始时间
 - `--timestop hh:mm[:ss]`: 结束时间
 - `[!] --monthdays day[,day...]`: 指定一个月的某一天
 - `[!] --weekdays day[,day...]`: 指定周一到周天
 - `--kerneltz`: 使用内核时区而不是UTC时间
- 示例：拒绝每天 8:30~12:30 (00:30~04:30)、
13:30~18:30 (05:30~10:30)，任何主机发送 icmp 协议；

```

# utc时间，与本地快8小时，所以需要-8小时
[root@Route ~]# iptables -t filter -I INPUT -p icmp -m time --timestampstart 00:30 --timestampstop 04:30 -j DROP
[root@Route ~]# iptables -t filter -I INPUT -p icmp -m time --timestampstart 05:30 --timestampstop 10:30 -j DROP

# 限制用户在上班时间段访问优酷、爱奇艺等资源，其他时间可以正常放行：
[root@route ~]# iptables -t filter -I OUTPUT -p tcp -m string --string "taobao.com" --algo kmp -m time --timestampstart 01:00 --timestampstop 04:00 -j DROP
[root@route ~]# iptables -t filter -I OUTPUT -p tcp -m string --string "taobao.com" --algo kmp -m time --timestampstart 06:00 --timestampstop 10:30 -j DROP
[root@route ~]# iptables -t filter -I OUTPUT -p tcp -m string --string "aqiyi.com" --algo kmp -m time --timestampstart 01:00 --timestampstop 04:00 -j DROP
[root@route ~]# iptables -t filter -I OUTPUT -p tcp -m string --string "aqiyi.com" --algo kmp -m time --timestampstart 06:00 --timestampstop 10:30 -j DROP

```

7.5.icmp模块

- `icmp` 扩展模块：默认情况当禁止 `ping` 后，其他主机无法 `ping` 通本主机，本主机也无法 `ping` 通其他主机。
- 假设现在的需求是本主机可以 `ping` 通其他主机，而其他主机依然无法 `ping` 同本主机。
- `[!] --icmp-type {type[/code] | typename}`
 - 指定 ICMP 类型，`echo-request`（8请求）、`echo-reply`（0回应）

```

# 常规做法不满足需求
[root@lb01 ~]# iptables -t filter -I INPUT -p icmp -j DROP

# 通过扩展 icmp
[root@lb01 ~]# iptables -t filter -F INPUT
[root@lb01 ~]# iptables -t filter -I INPUT -p icmp --icmp-type "echo-request" -j REJECT
[root@lb01 ~]# iptables -L -n
Chain INPUT (policy ACCEPT)
target      prot opt source          destination
REJECT      icmp --  0.0.0.0/0        0.0.0.0/0
icmp type 8 reject-with icmp-port-unreachable

```

7.7.connlimit模块

- `connlimit` 扩展模块，限制每个客户端IP地址到服务器的并行连接数。
 - `--connlimit-up to n`: 如果现有连接数小于或等于n，则匹配。
 - `--connlimit-above n`: 如果现有连接数大于n，则匹配。
- 脚本程序flood.c，下载地址：[flood_connect.c](#)
- 示例：限制，每个客户端主机允许2个telnet连接。

```
[root@lb01 ~]# yum install telnet-server -y
[root@lb01 ~]# systemctl start telnet.socket

[root@lb01 ~]# iptables -A INPUT -p tcp --dport 23 -m connlimit \
-connlimit-above 2 -j REJECT
[root@lb01 ~]# iptables -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    REJECT     tcp   --  0.0.0.0/0      0.0.0.0/0
tcp dpt:23 #conn src/32 > 2 reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
num  target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
```

- 示例：限制，每个客户端主机允许100并发连接80/tcp

```
[root@lb01 ~]# iptables -I INPUT -p tcp --syn --dport 80 -m \
connlimit --connlimit-above 100 -j DROP
```

7.6.limit模块

- limit扩展模块：针对“报文速率”进行限制。
- 限制单位时间内流入包的数量，我们可以以秒为单位进行限制，也可以以分钟、小时、天作为单位进行限制。
 - `--limit rate[second|minute|hour|day]`: 平均匹配的速率
 - `--limit-burst number`: 超过限制速率的包，允许超过burst所设定值，默认可超出5个

1. 限制主机每分钟接收10个icmp数据包，差不多6s会接收客户端一个数据包。

```
[root@lb01 ~]# iptables -t filter -F
[root@lb01 ~]# iptables -t filter -I INPUT -p icmp -m limit --limit 10/minute -j ACCEPT
[root@lb01 ~]# iptables -t filter -A INPUT -p icmp -j REJECT
```

2.允许icmp瞬间通过10个数据包通过，超过的数据包每分钟仅能通过一个。

```
[root@lb01 ~]# iptables -A INPUT -p icmp -m limit --limit 1/m --limit-burst 10 -j ACCEPT
[root@lb01 ~]# iptables -A INPUT -p icmp -j REJECT # 如果超过10的我们给其drop掉
```

3.限制主机传输时的带宽每秒不超过 500k；
 $(500k * 1000 = 500000 \text{字节} / 1500 = 333 \text{个包})$ [单位换算网站](#)

```
[root@lb01 ~]# iptables -t filter -I OUTPUT -p tcp -m limit --limit 300/second -j ACCEPT
[root@lb01 ~]# iptables -t filter -A OUTPUT -p tcp -j DROP
```

7.6.tcp-flags模块

- 使用tcp扩展模块的"--tcp-flags"选项，即可对TCP的标志位进行匹配，匹配指定标志位的值是否为"1"，在tcp协议建立连接的过程中，需要先进行三次握手，而三次握手就要依靠tcp头中的标志位进行。

1.第一次：客户端向服务端发起第一次TCP连接时，在TCP的flag标志位中，SYN,RST,ACK,FIN等标志位仅有SYN为1，其他标志位为0。

```
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.8
▼ Transmission Control Protocol, Src Port: 60696, Dst Port: 22, Seq: 0, Len: 0
  Source Port: 60696
  Destination Port: 22
  [Stream index: 1]
  [TCP Segment Len: 0]
  Sequence number: 0      (relative sequence number)
  [Next sequence number: 0      (relative sequence number)]
  Acknowledgment number: 0
  1011 .... = Header Length: 44 bytes (11)
▼ Flags: 0x002 (SYN)
  000. .... .... = Reserved: Not set
  ....0 .... .... = Nonce: Not set
  .... 0.... .... = Congestion Window Reduced (CWR): Not set
  .... .0.... .... = ECN-Echo: Not set
  .... ..0.... .... = Urgent: Not set
  .... ...0.... .... = Acknowledgment: Not set
  .... ....0... .... = Push: Not set
  .... .... .0.. .... = Reset: Not set
  .... .... ..1. .... = Syn: Set
  .... .... ..0.... .... = Fin: Not set
  [TCP Flags: .... ..S.]
```

第二次：服务端向客户端返回ACK，在TCP的flag标志位中，SYN,RST,ACK,FIN等标志位SYN、ACK为1，其他标志位为0。

```

> Internet Protocol Version 4, Src: 10.0.0.8, Dst: 10.0.0.1
▼ Transmission Control Protocol, Src Port: 22, Dst Port: 60696, Seq: 0, Ack: 1, Len: 0
  Source Port: 22
  Destination Port: 60696
  [Stream index: 1]
  [TCP Segment Len: 0]
  Sequence number: 0      (relative sequence number)
  [Next sequence number: 0      (relative sequence number)]
  Acknowledgment number: 1      (relative ack number)
  1010 .... = Header Length: 40 bytes (10)
  ▼ Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0.... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    ▶ .... .... ..1. = Syn: Set
    .... .... ..0 = Fin: Not set
    [TCP Flags: .....A..S..]

```

第三次：客户端向服务端返回ACK，在TCP的flag标志位中，SYN,RST,ACK,FIN等标志位ACK为1，其他标志位为0。

```

> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.8
▼ Transmission Control Protocol, Src Port: 60696, Dst Port: 22, Seq: 1, Ack: 1, Len: 0
  Source Port: 60696
  Destination Port: 22
  [Stream index: 1]
  [TCP Segment Len: 0]
  Sequence number: 1      (relative sequence number)
  [Next sequence number: 1      (relative sequence number)]
  Acknowledgment number: 1      (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  ▼ Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0.... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: .....A....]

```

- 我们可以通过`--tcp-flag`指明需要匹配哪些标志位，然后再指明这些标志位中，哪些必须为1，剩余的都必须为0。
- 所以当服务器接收新请求时，SYN标志位必须1，其他的标志位为0，
- 而服务端响应这个连接时，SYN、ACK标志位必须为1，其他的标志位为0。（这样可以避免木马程序通过端口主动向外发送新连接。）

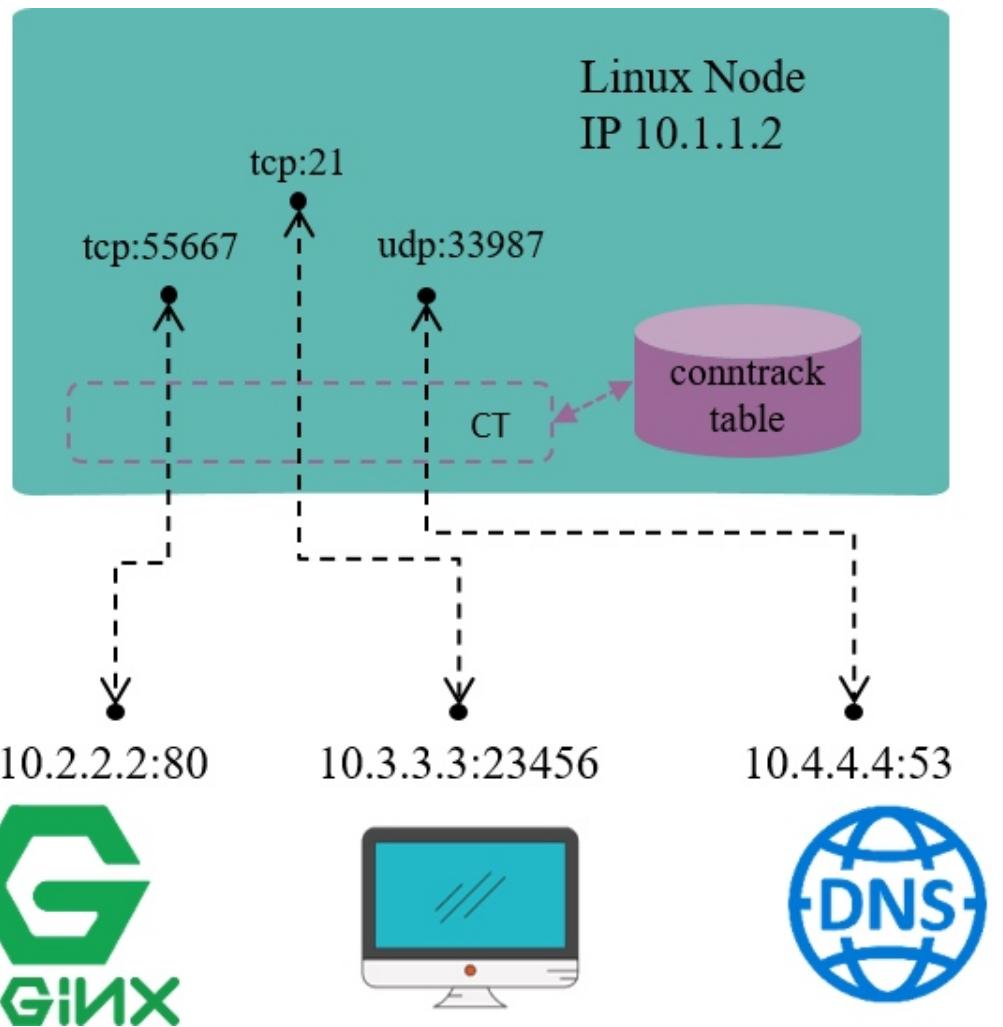
示例：客户端连接服务端22端口第一次握手必须是客户端发起的，所以SYN必须为1，剩下全部为0。然后服务端可以通过22端口返回对应的报文。

```
# 使用"--syn"选项相当于使用"--tcp-flags SYN,RST,ACK,FIN  SYN"
[root@lb01 ~]# iptables -t filter -I INPUT -p tcp -m tcp --dport 22 --tcp-flags SYN,ACK,FIN,RST SYN -j ACCEPT
[root@lb01 ~]# iptables -t filter -A INPUT -p tcp -m tcp --dport 22 --tcp-flags SYN,ACK,FIN,RST ACK -j ACCEPT
[root@lb01 ~]# iptables -t filter -A INPUT -j DROP

# output
[root@lb01 ~]# iptables -t filter -I OUTPUT -p tcp --sport 22 -m tcp --tcp-flags SYN,ACK,FIN,RST SYN,ACK -j ACCEPT
[root@lb01 ~]# iptables -t filter -A OUTPUT -p tcp --sport 22 -m tcp --tcp-flags SYN,ACK,FIN,RST ACK -j ACCEPT
[root@lb01 ~]# iptables -t filter -A OUTPUT -j DROP
```

7.8.state模块

- `state (conntrack)` 连接跟踪，顾名思义，就是跟踪（并记录）连接的状态。
 - 如下图：是一台 IP 地址为 10.1.1.2 的 Linux 机器，我们能看到这台机器上有三条连接：
 - 机器访问外部 HTTP 服务的连接（目的端口 80）
 - 外部访问机器内 FTP 服务的连接（目的端口 21）
 - 机器访问外部 DNS 服务的连接（目的端口 53）



连接跟踪所做的事情就是发现并跟踪这些连接的状态，但这个追踪状态与TCP协议没有关系。它是由内核nefiter在IP层实现，可IP层是无连接、无追踪的，那是如何知道这个IP此前到底有没有来过？

当用户发送请求时，会将用户的请求存储在内核开辟的内存空间中，记录源IP、目标IP、协议、时间，当下次在有用户发起请求，就可以通过内存空间中获取该用户是否来过，以此来实现连接追踪机制。那连接追踪的状态有哪些：

[New]：新请求，内存中不存在此连接的相关条目，因此识别为第一次请求，状态为NEW

[ESTABLISHED]：NEW状态之后，再次建立连接，由于此前的连接还没有失效，所以追踪后被视为已连接通讯状态，状态为ESTABLISHED

[RELATED]：相关的连接。比如ftp有两个连接，命令连接和数据连接。命令连接有来有往是一个独立的循环，数据连接有来有往又是另外一个独立的循环，但是两者之间有关系，如果没有命令连接就不可能有数据连接，所以我们将这种称为“相关联的连接”

[INVALID]：无效的连接。

应用场景举例：

正常情况下服务器的80端口不会主动连接其他服务器，如果出现了80端口连接其他服务器，那么说明出现了异常行为，或者可以理解为中了木马程序病毒。[反弹端口型木马](#)

如果关闭80端口的响应报文，但那样就会造成请求进来无法响应；如果开放80端口，则又会出现异常行为。

所以我们需要从80端口做连接追踪限制，凡事从80端口出去的就必须是对某个请求的响应，也就是说通过80端口出去的状态必须是ESTABLISHED，不能是NEW。

示例1：允许接收远程主机的SSH与HTTP请求（NEW、ESTABLISHED），同时仅允许本机回应SSH以及HTTP的响应（ESTABLISHED），但不允许本机通过22、80端口主动向外发起连接。

```
[root@lb01 ~]# iptables -I INPUT -p tcp -m multiport --dport 80,22 -m state --state NEW,ESTABLISHED -j ACCEPT
[root@lb01 ~]# iptables -I OUTPUT -p tcp -m multiport --sport 80,22 -m state --state ESTABLISHED -j ACCEPT
[root@lb01 ~]# iptables -A INPUT -j DROP
[root@lb01 ~]# iptables -A OUTPUT -j DROP

[root@lb01 ~]# iptables -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0
multiport dports 80,22 state NEW,ESTABLISHED
2    DROP       all  --  0.0.0.0/0        0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
1    ACCEPT     tcp  --  0.0.0.0/0        0.0.0.0/0
multiport sports 80,22 state ESTABLISHED
2    DROP       all  --  0.0.0.0/0        0.0.0.0/0
```

示例2.如果服务器也需要使用SSH连接其他远程主机，则还需要增加以下配置（但不建议）

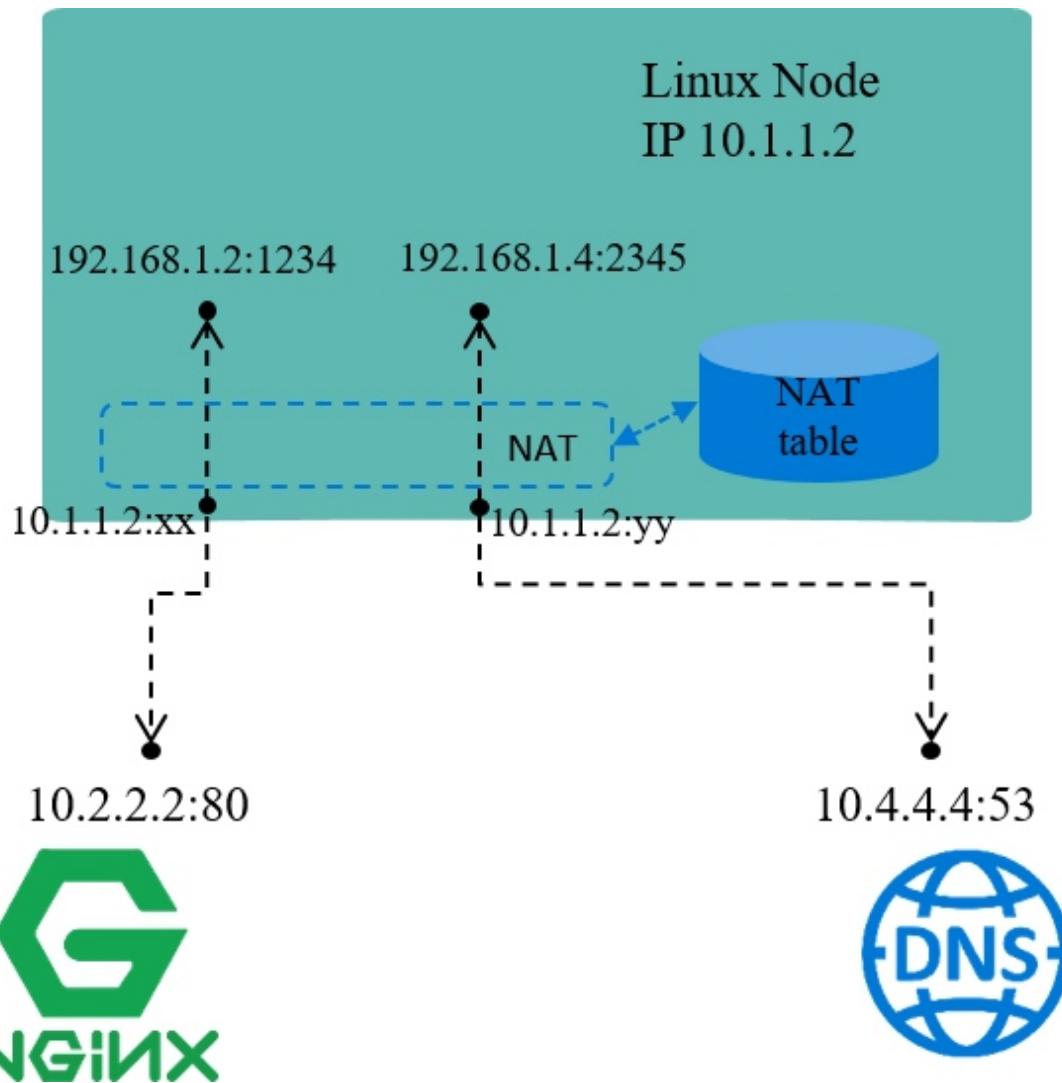
```
# 1.送出的数据包目的端口为22
[root@lb01 ~]# iptables -I OUTPUT 2 -p tcp --dport 22 -m state --
state NEW,ESTABLISHED -j ACCEPT

# 2.接收的数据包源端口为22
[root@lb01 ~]# iptables -I INPUT 2 -p tcp --sport 22 -m state --
state ESTABLISHED -j ACCEPT
```

8.iptables地址转换

8.1.什么是NAT

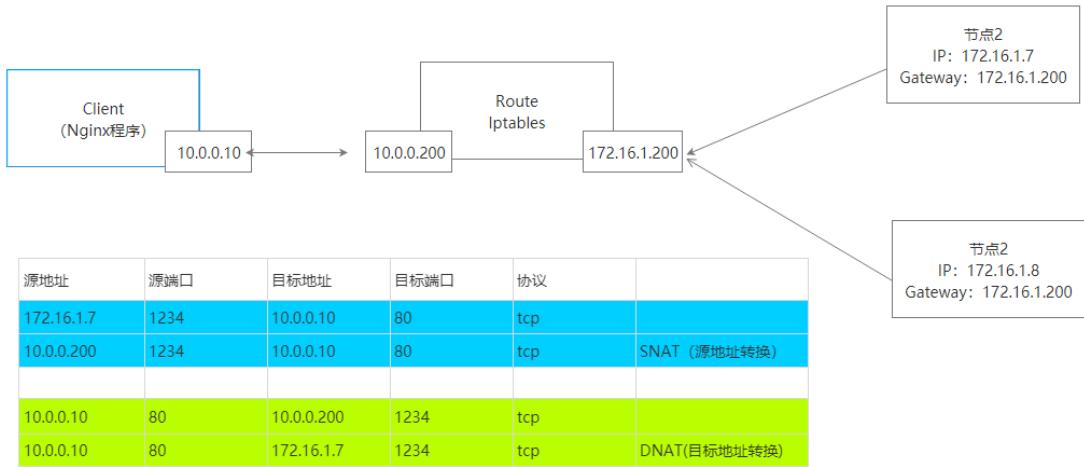
- 网络地址转换（NAT），意思也比较清楚：对（数据包的）网络地址（IP + Port）进行转换。
- 例如，机器自己的 IP 10.1.1.2 是能与外部正常通信的，但 192.168 网段是私有 IP 段，无法与外界通信，因此当源地址为 192.168 网段的包要出去时，机器会先将源 IP 换成机器自己的 10.1.1.2 再发送出去；收到应答包时，再进行相反的转换。这就是 NAT 的基本过程。



8.2 NAT的几种模式

- SNAT：源地址转换
- DNAT：目标地址转换
- PNAT：端口转换

8.3 NAT环境搭建



8.3.SNAT场景示例

场景：实现内网主机通过防火墙进行上网，需要使用SNAT（源地址转换 POSTROUTING）

172.16.1.7 -->172.16.1.5 --POSTROUTING-->10.0.0.5--> baidu.com

1. 将172.16.1.7的网关指向172.16.1.5

```
[root@web02 ~]# echo "GATEWAY=172.16.1.5" >> /etc/sysconfig/network-scripts/ifcfg-eth1
[root@web02 ~]# ifdown eth1 && ifup eth1
```

2. 开启iptables防火墙的forward转发

```
[root@lb01 ~]# echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
[root@lb01 ~]# sysctl -p
```

3. 配置iptables的SNAT转发规则

```
# 方式一：指定从哪个ip地址转换出去（静态公网地址）
[root@route ~]# iptables -t nat -I POSTROUTING -s 172.16.1.0/24 -j SNAT --to 10.0.0.200
[root@route ~]# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target      prot opt source          destination
Chain INPUT (policy ACCEPT)
target      prot opt source          destination
Chain OUTPUT (policy ACCEPT)
target      prot opt source          destination
Chain POSTROUTING (policy ACCEPT)
```

```
target      prot opt source          destination
SNAT       all   --  172.16.1.0/24    0.0.0.0/0
to:10.0.0.200
```

方式二：当外网源地址为动态获取的地址时，MASQUERADE可自行判断要转换为的外网地址

```
# iptables -t nat -A POSTROUTING -s 172.16.1.0/24 -j MASQUERADE
```

8.4.DNAT场景示例

2. 实现外网主机通过防火墙访问内部主机80端口，需要通过DNAT（目标地址转换PREROUTING）

```
# 端口映射
[root@route ~]# iptables -t nat -I PREROUTING -d 10.0.0.200 -p
tcp --dport 80 -j DNAT --to 172.16.1.7:80
[root@route ~]# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target      prot opt source          destination
DNAT       tcp   --  0.0.0.0/0      10.0.0.200           tcp
dpt:80 to:172.16.1.7:80

# 地址映射
[root@route ~]# iptables -t nat -I PREROUTING -d 10.0.0.200 -j
DNAT --to 172.16.1.7
```

9.Iptables自定义链

9.1.为什么要使用自定义链

此前我们一直在iptables的默认链中定义规则，那么此处，我们就来了解一下自定义链。这里可能有疑问，iptables的默认链就已经能够满足我们了，为什么还需要自定义链呢？

当默认链中的规则非常多时，不便于管理。假设INPUT链中存放了200条规则，这200条规则有针对80端口的，有针对22端口的，有针对私网IP的，等等。

假如想修改针对80端口的规则，可能需要从头到尾看一遍200条规则，找出哪些规则是针对80端口。这显然不合理。所以，我们需要使用自定义链，通过自定义链即可解决上述问题。

假设，我们自定义一条链，链名叫 IN_HTTP，我们可以将所有针对80端口入站规则都写入到这条自定义链中，当以后想要修改针对80端口入站规则时，就直接修改 IN_HTTP 链中的规则就可以了。

即使默认链中有再多的规则，也没有关系，因为所有针对80端口的入站规则都存放在 IN_HTTP 链中。同理，我们可以将针对22端口的出站规则放入到OUT_SSH自定义链中，这样，我们就能快速定位到想修改的规则在哪里。

9.2.自定义链基本应用

1.在 filter 表中，添加一个自定义链

```
[root@lb01 ~]# iptables -t filter -N IN_HTTP
[root@lb01 ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Chain IN_HTTP (0 references)           # 自定义的链、无法直接使用
target     prot opt source               destination
```

2.给自定义链配置对应的规则。禁止 10.0.0.10 访问 10.0.0.200 的 22 端口。

```
[root@lb01 ~]# iptables -t filter -I IN_HTTP -s 10.0.0.10 -d
10.0.0.200 -p tcp --dport 22 -j REJECT
[root@lb01 ~]# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Chain IN_HTTP (0 references)
target     prot opt source               destination
REJECT    tcp   --  10.0.0.1            10.0.0.5          tcp
dpt:80  reject-with icmp-port-unreachable
```

3. 测试后发现，规则没有生效，因为自定义的链需要被默认的链调用才会生效。既然 IN_HTTP 链是为了针对 22 端口的入站规则而创建的，那么这些规则应该去匹配入站的报文，所以应该用 INPUT 链去引用它。

```
[root@lb01 ~]# iptables -I INPUT -p tcp --dport 22 -j IN_HTTP
[root@lb01 ~]# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
IN_HTTP    tcp   --  0.0.0.0/0          0.0.0.0/0           tcp
dpt:80

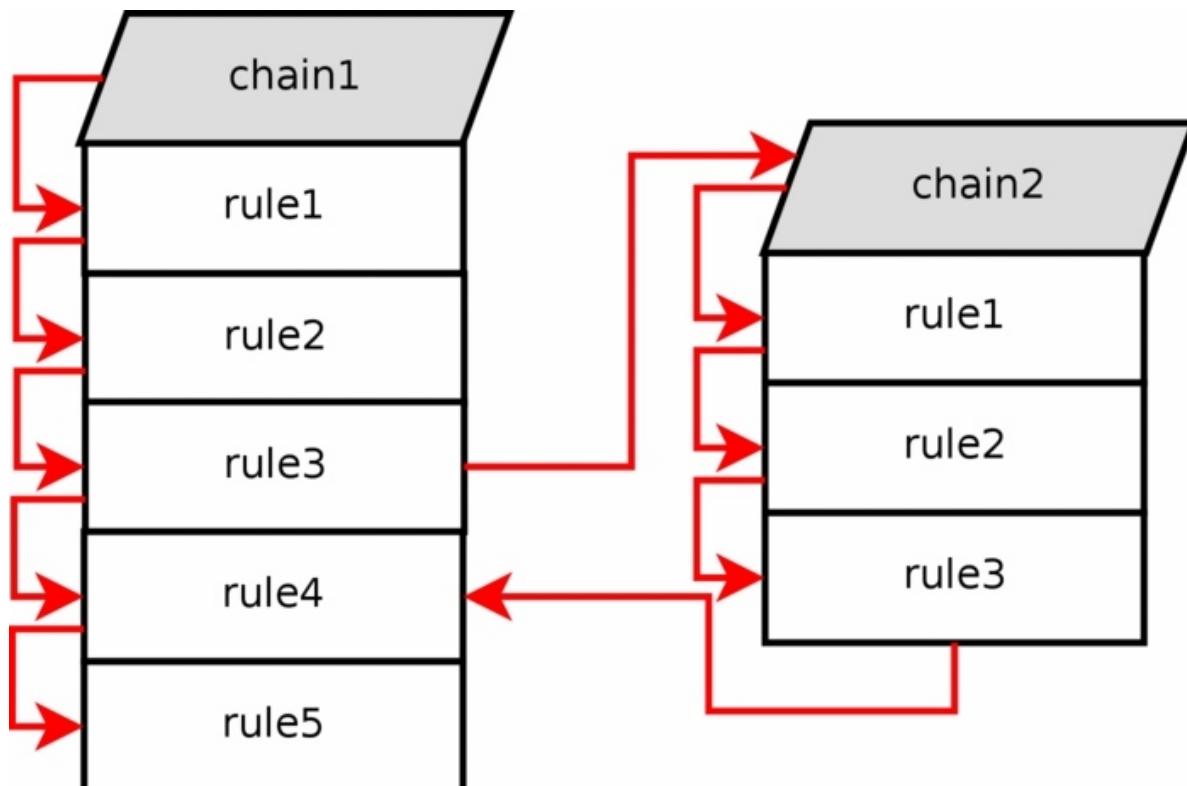
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain IN_HTTP (1 references)
target     prot opt source               destination
REJECT    tcp   --  10.0.0.1           10.0.0.5          tcp
dpt:80  reject-with icmp-port-unreachable
```

4. 测试后发现 10.0.0.10，无法访问 22 端口，原因请求本机 22 经过 INPUT 链，而 INPUT 链将 22 端口处理规则转给 IN_HTTP 链，由 IN_HTTP 链进行规则匹配，匹配源 IP 是 10.0.0.1 则拒绝。而其他主机则会匹配规则失败，那么会继续匹配后续的规则，如果没有规则匹配，则回到默认 INPUT 链，继续往下匹配规则，如果没有匹配成功的规则，则走 INPUT 的默认规则。

9.3. 自定义链执行顺序



9.4.如何删除自定义链

删除自定义链需要满足两个条件：1.自定义链没有被引用、2.自定义链中没有任何规则

```

[root@lb01 ~]# iptables -t filter -D IN_HTTP 1      # 删除自定义规则
[root@lb01 ~]# iptables -t filter -D INPUT 1       # 删除INPUT引用
[root@lb01 ~]# iptables -E IN_HTTP HTTP           # 自定义链重命令
[root@lb01 ~]# iptables -X HTTP                  # 删除自定义链

```

02.SSH远程管理服务实战

1.SSH基本概述

1.什么是SSH

`SSH` 是一个安全协议，在进行数据传输时，会对数据包进行加密处理，加密后在进行数据传输。确保了数据传输安全。

2.SSH服务主要提供什么功能

- 1.提供远程连接服务器的服务
- 2.对远程连接传输数据进行加密

3.SSH远程连接 VS Telnet远程连接的区别?

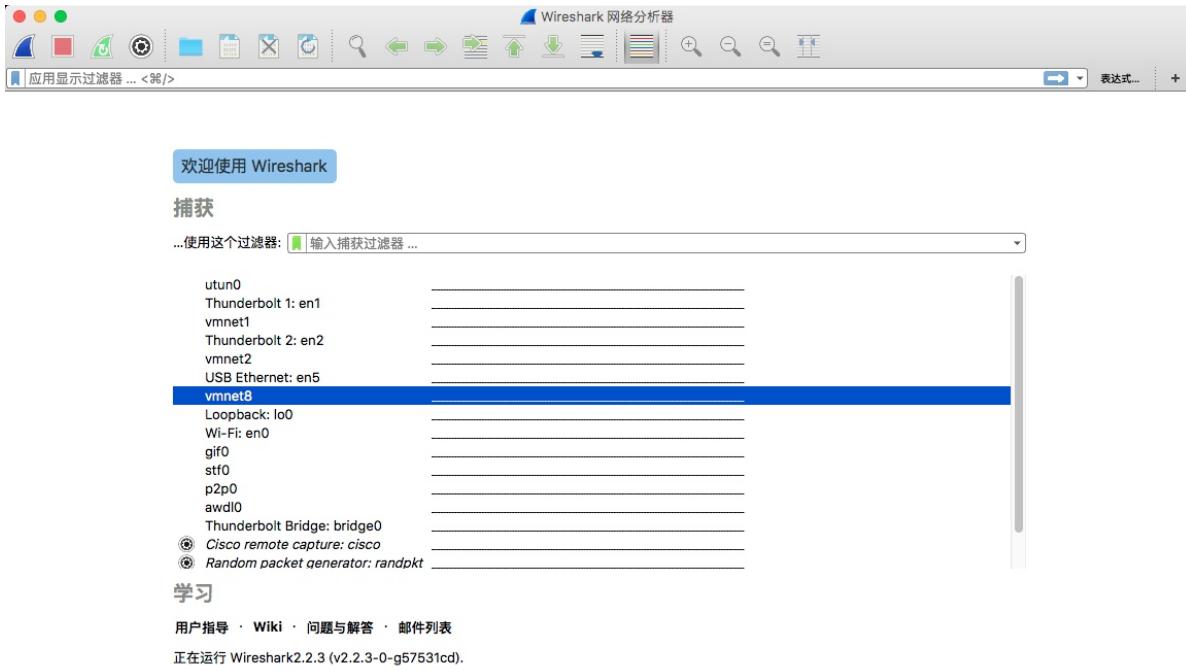
服务连接方式	服务数据传输	服务监听端口	服务登陆用户
ssh	加密	22/tcp	默认支持root用户登陆
telnet	明文	23/tcp	不支持root用户登陆

4.案例: 使用wireshark 抓包验证telnet 明文传输与ssh 密文传输

1.安装telnet服务并运行

```
[root@m01 ~]# yum install telnet-server -y  
[root@m01 ~]# systemctl start telnet.socket
```

2.使用wireshark检测vmnet8网卡上telnet的流量



3.telnet是无法使用root用户登录Linux系统，需要创建普通用户

```
[root@m01 ~]# useradd bgx  
[root@m01 ~]# echo "123456" | passwd --stdin bgx
```

4.使用普通用户进行telnet登录

```
Default
telnet 10.0.0.40
Trying 10.0.0.40...
Connected to bogon.
Escape character is '^]'.
Password:
Login incorrect

m01 Login: bgx
Password:
Last Login: Sun Jul 29 07:38:06 from bogon
[bgx@m01 ~]$
```

5.搜索 wireshark 包含 telnet 相关的流量

Wireshark - telnet (vmnet2)

No.	Time	Source	Destination	Protocol	Length	Info
35	10.762455	10.0.0.1	10.0.0.40	TELNET	68	Telnet Data ...
39	10.764221	10.0.0.40	10.0.0.1	TELNET	68	Telnet Data ...
47	13.659553	10.0.0.40	10.0.0.1	TELNET	107	Telnet Data ...
49	14.658728	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...
50	14.659040	10.0.0.40	10.0.0.1	TELNET	67	Telnet Data ...
52	14.881862	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...
53	14.883560	10.0.0.40	10.0.0.1	TELNET	67	Telnet Data ...
55	15.013414	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...
56	15.013650	10.0.0.40	10.0.0.1	TELNET	67	Telnet Data ...
58	15.112948	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...
59	15.113150	10.0.0.40	10.0.0.1	TELNET	67	Telnet Data ...
63	15.629284	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...
64	15.629531	10.0.0.40	10.0.0.1	TELNET	69	Telnet Data ...
66	15.730011	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...
67	15.730212	10.0.0.40	10.0.0.1	TELNET	69	Telnet Data ...
69	15.852046	10.0.0.1	10.0.0.40	TELNET	67	Telnet Data ...

Frame 47: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
Ethernet II, Src: VMware_fd:ed:c6 (00:0c:29:fd:ed:c6), Dst: VMware_c0:00:02 (00:50:56:c0:00:02)
Internet Protocol Version 4, Src: 10.0.0.40, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 23, Dst Port: 49364, Seq: 76, Ack: 184, Len: 41
Telnet
Data: Login incorrect\r\n
Data: \r\n
Data: ansible-node40 login:
0010 00 5d ae 44 40 00 40 06 78 1e 0a 00 00 28 0a 00 .].D@.x....(..
0020 00 01 00 17 c0 d4 a9 17 ba b3 24 01 f4 a8 00 18\$.....
0030 00 eb d5 ea 00 00 01 01 08 0a 00 3b 22 c9 4a d7;"..J.
0040 b8 b7 4c 6f 67 69 6e 20 69 6e 63 6f 72 72 65 63 ..Login incorre
0050 74 0d 0a 0d 0a 61 6e 73 69 62 6c 65 2d 6e 6f 64 t...ans ible-nod
0060 65 34 30 20 6c 6f 67 69 6e 3a 20 e40 logi n:
Data (telnet.data), 17 bytes

Wireshark - Data (telnet.data), 17 bytes (vmnet2)

No.	Time	Source	Destination	Protocol	Length	Info
35	10.762455	10.0.0.1	10.0.0.40	TELNET	68	Telnet Data ...
39	10.764221	10.0.0.40	10.0.0.1	TELNET	68	Telnet Data ...
47	13.659553	10.0.0.40	10.0.0.1	标记/取消标记 分组(M)	68	Telnet Data ...
49	14.658728	10.0.0.1	10.0.0.40	忽略/取消忽略 分组(I)	67	Telnet Data ...
50	14.659040	10.0.0.40	10.0.0.1	设置/取消设置 时间参考	67	Telnet Data ...
52	14.881862	10.0.0.1	10.0.0.40	时间平移...	67	Telnet Data ...
53	14.883560	10.0.0.40	10.0.0.1	分组注释...	67	Telnet Data ...
55	15.013414	10.0.0.1	10.0.0.40	编辑解析的名称	67	Telnet Data ...
56	15.013650	10.0.0.40	10.0.0.1	作为过滤器应用	67	Telnet Data ...
58	15.112948	10.0.0.1	10.0.0.40	准备过滤器	67	Telnet Data ...
59	15.113150	10.0.0.40	10.0.0.1	对话过滤器	67	Telnet Data ...
63	15.629284	10.0.0.1	10.0.0.40	对话着色	67	Telnet Data ...
64	15.629531	10.0.0.40	10.0.0.1	SCTP	67	Telnet Data ...
66	15.730011	10.0.0.1	10.0.0.40	追踪流	67	TCP 流 data ...
67	15.730212	10.0.0.40	10.0.0.1	复制	67	UDP 流 data ...
69	15.852046	10.0.0.1	10.0.0.40	协议首选项	67	SSL 流 face 0

Frame 47: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
Ethernet II, Src: VMware_fd:ed:c6 (00:0c:29:fd:ed:c6), Dst: VMware_c0:00:02 (00:50:56:c0:00:02)
Internet Protocol Version 4, Src: 10.0.0.40, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 23, Dst Port: 49364, Seq: 76, Ack: 184, Len: 41
Telnet
Data: Login incorrect\r\n
Data: \r\n
Data: ansible-node40 login:
0010 00 5d ae 44 40 00 40 06 78 1e 0a 00 00 28 0a 00 .].D@.x....(..
0020 00 01 00 17 c0 d4 a9 17 ba b3 24 01 f4 a8 00 18\$.....
0030 00 eb d5 ea 00 00 01 01 08 0a 00 3b 22 c9 4a d7;"..J.
0040 b8 b7 4c 6f 67 69 6e 20 69 6e 63 6f 72 72 65 63 ..Login incorre
0050 74 0d 0a 0d 0a 61 6e 73 69 62 6c 65 2d 6e 6f 64 t...ans ible-nod
0060 65 34 30 20 6c 6f 67 69 6e 3a 20 e40 logi n:
Data (telnet.data), 17 bytes

Wireshark - 追踪 TCP 流 (tcp.stream eq 0) · wireshark_vmnnet2_20180729193825_WyGm5s

```
..%..... !..#..... #..... #.'..%. ....!..%.....  

.....#.....'.....I..... .38400,38400....#.DISPLAY./private/tmp/  

com.apple.launchd.hhwKxGHLIN/org.macossforg.xquartz:  

0.USER.xuliangwei.....XTERM-256COLOR....#.....Password:  

.  

Login incorrect  

m01 login: bbggxx  

.  

Password: 123456  

.  

Last login: Sun Jul 29 07:38:06 from bogon  

.]0;bgx@m01:~..[?1034h[bgx@m01 ~]$
```



6. 使用wireshark分析ssh流量

```

Default
ssh bgx@10.0.0.40
bgx@10.0.0.40's password:
Last Login: Sun Jul 29 07:38:35 2018 from bogon
[bgx@m01 ~]$ 

```

ssh

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000773	10.0.0.1	10.0.0.40	SSHv2	87	Client: Protocol (SSH-2.0-OpenSSH_7.3)
6	0.014144	10.0.0.40	10.0.0.1	SSHv2	87	Server: Protocol (SSH-2.0-OpenSSH_7.4)
8	0.015583	10.0.0.1	10.0.0.40	SSHv2	14...	Client: Key Exchange Init
9	0.016748	10.0.0.40	10.0.0.1	SSHv2	13...	Server: Key Exchange Init
11	0.019675	10.0.0.1	10.0.0.40	SSHv2	114	Client: Diffie-Hellman Key Exchange Init
12	0.026967	10.0.0.40	10.0.0.1	SSHv2	430	Server: Diffie-Hellman Key Exchange Reply, New Keys...
14	0.034203	10.0.0.1	10.0.0.40	SSHv2	82	Client: New Keys
16	0.074557	10.0.0.1	10.0.0.40	SSHv2	110	Client: Encrypted packet (len=44)
18	0.074794	10.0.0.40	10.0.0.1	SSHv2	110	Server: Encrypted packet (len=44)
20	0.074883	10.0.0.1	10.0.0.40	SSHv2	126	Client: Encrypted packet (len=60)
25	0.085503	10.0.0.40	10.0.0.1	SSHv2	150	Server: Encrypted packet (len=84)
27	0.085647	10.0.0.1	10.0.0.40	SSHv2	430	Client: Encrypted packet (len=364)
28	0.088124	10.0.0.40	10.0.0.1	SSHv2	150	Server: Encrypted packet (len=84)
30	2.528154	10.0.0.1	10.0.0.40	SSHv2	150	Client: Encrypted packet (len=84)
31	2.547141	10.0.0.40	10.0.0.1	SSHv2	94	Server: Encrypted packet (len=28)
33	2.548326	10.0.0.1	10.0.0.40	SSHv2	178	Client: Encrypted packet (len=112)

Frame 4: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
Ethernet II, Src: Vmware_c0:00:02 (00:50:56:c0:00:02), Dst: Vmware_fd:ed:c6 (00:0c:29:fd:ed:c6)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.40
Transmission Control Protocol, Src Port: 53858, Dst Port: 22, Seq: 1, Ack: 1, Len: 21
SSH Protocol
Protocol: SSH-2.0-OpenSSH_7.3\r\nn

0000 00 0c 29 fd ed c6 00 50 56 c0 00 02 08 00 45 02 ..)...P V.....E.
0010 00 49 dd b1 40 00 40 06 48 d3 0a 00 00 01 0a 00 .I..@. H.....
0020 00 28 d2 62 00 16 78 4f 0d 86 2b 27 2c 6e 80 18 .(b..x0 ..+,'n..
0030 10 15 ee 8e 00 00 01 01 08 0a 4c a6 34 83 00 08L.4...
0040 70 16 53 53 48 2d 32 2e 30 2d 4f 70 65 6e 53 53 p,SSH-2. 0-OpenSS
0050 48 5f 37 2e 33 0d 0a H_7.3..

SSH Protocol: Protocol

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000773	10.0.0.1	10.0.0.40	SSHv2	87	Client: Protocol (SSH-2.0-OpenSSH_7.3)
5	0.001033	10.0.0.1	10.0.0.40	TCP	66	22-53858 [ACK] Seq=1 Ack=22 Win=29056 Len=0 TSval=5...
6	0.014144	10.0.0.1	10.0.0.40	SSHv2	87	Server: Protocol (SSH-2.0-OpenSSH_7.4)
7	0.014197	10.0.0.1	10.0.0.40	TCP	66	53858-22 [ACK] Seq=22 Ack=22 Win=131744 Len=0 TSval...
8	0.015583	10.0.0.1	10.0.0.40	SSHv2	14...	Client: Key Exchange Init
9	0.016748	10.0.0.1	10.0.0.40	SSHv2	13...	Server: Key Exchange Init
10	0.016805	10.0.0.1	10.0.0.40	TCP	66	53858-22 [ACK] Seq=1454 Ack=1302 Win=130464 Len=0 T...
11	0.019675	10.0.0.1	10.0.0.40	SSHv2	114	Client: Diffie-Hellman Key Exchange Init
12	0.026967	10.0.0.1	10.0.0.40	SSHv2	430	Server: Diffie-Hellman Key Exchange Reply, New Keys...
13	0.027026	10.0.0.1	10.0.0.40	TCP	66	53858-22 [ACK] Seq=1502 Ack=1666 Win=130688 Len=0 T...
14	0.034203	10.0.0.1	10.0.0.40	SSHv2	82	Client: New Keys
15	0.074513	10.0.0.1	10.0.0.40	TCP	66	22-53858 [ACK] Seq=1666 Ack=1518 Win=31872 Len=0 TS...
16	0.074557	10.0.0.1	10.0.0.40	TCP 流	110	Client: Encrypted packet (len=44)
17	0.074670	10.0.0.1	10.0.0.40	UDP 流	66	22-53858 [ACK] Seq=1666 Ack=1562 Win=31872 Len=0 TS...
18	0.074794	10.0.0.1	10.0.0.40	SSL 流	110	Server: Encrypted packet (len=44)
19	0.074819	10.0.0.1	10.0.0.40	HTTP 流	66	53858-22 [ACK] Seq=1562 Ack=1710 Win=131008 Len=0 T...

Frame 4: 87 bytes on wire (696 bits) on interface 0
Ethernet II, Src: Vmware_c0:00:02 (00:50:56:c0:00:02), Dst: Vmware_fd:ed:c6 (00:0c:29:fd:ed:c6)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.40
Transmission Control Protocol, Src Port: 53858, Dst Port: 22, Seq: 1, Ack: 1, Len: 21
SSH Protocol
Protocol: SSH-2.0-OpenSSH_7.3\r\nn

Internet Protocol Version 4 (ip, 20 字节)

Wireshark - 追踪 TCP 流 (tcp.stream eq 0) · wireshark_vmnet2_20180729194055_OsJT47

SSH-2.0-OpenSSH_7.3

SSH-2.0-OpenSSH_7.4

.....R....D....?.....curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c..." "ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,ssh-ed25519,rsa-sha2-512,rsa-sha2-256,ssh-rsa....chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,3des-cbc....chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,3des-cbc....umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etc...etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1....umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etc...etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1....none,zlib@openssh.com,zlib....none,zlib@openssh.com,zlib.....

...
...e.[_g..T.....@curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1...Assh-rsa,rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed25519....chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,blowfish-cbc,cast128-cbc,3des-cbc....chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,aes128-cbc,aes192-cbc,aes256-cbc,blowfish-cbc,cast128-cbc,3des-cbc.....umac-64-etm@openssh.com,umac-128

分组 8。27 客户端 分组, 29 服务器 分组, 49 turn(s).点击选择。

Entire conversation (6986 bytes) 显示和保存数据为 ASCII 流 0

查找: 查找下一个(N)

Help 滤掉此流 打印 Save as... Back Close

2.SSH客户端命令

SSH有客户端与服务端，我们将这种模式称为C/S架构，ssh客户端支持Windows、Linux、Mac等平台。

在ssh客户端中包含 `ssh|slogin` 远程登陆、`scp` 远程拷贝、`sftp` 文件传输、`ssh-copy-id` 秘钥分发等应用程序。

2.1 ssh远程登陆

1.ssh远程登录服务器命令示例

```
ssh -p22 root@10.0.0.61
```

```
# -p指定连接远程主机端口，默认22端口可省略
# root@remotehost
# "@"前面为用户名，如果用当前用户连接，可以不指定用户
# "@"后面为要连接的服务器的IP
```

2.2 scp远程拷贝

scp 复制数据至远程主机命令(全量复制)

-P 指定端口， 默认22端口可不写

-r 表示递归拷贝目录

-p 表示在拷贝文件前后保持文件或目录属性不变

-l 限制传输使用带宽(默认kb)

#推：将本地/tmp/oldxu推送至远端服务器10.0.0.61的/tmp目录，使用对端的root用户

```
[root@m01 ~]# scp -P22 -rp /tmp/oldxu root@10.0.0.61:/tmp
```

#拉：将远程10.0.0.61服务器/tmp/file文件拉取到本地/opt/目录下

```
[root@m01 ~]# scp -P22 -rp root@10.0.0.61:/tmp/file /opt/
```

#限速

```
[root@m01 ~]# scp /opt/1.txt root@172.16.1.31:/tmp
```

root@172.16.1.31 password:

```
test 100% 656MB '83.9MB/s' 00:07
```

#限速为8096kb，换算为MB，要除以 8096/8=1024KB=1MB

```
[root@m01 ~]# scp -rp -l 8096 /opt/1.txt root@172.16.1.31:/tmp
```

root@172.16.1.31s password:

```
test 7% 48MB '1.0MB/s' 09:45
```

- 注意：

- 1.scp通过ssh协议加密方式进行文件或目录拷贝。
- 2.scp连接时的用户作为为拷贝文件或目录的权限。
- 3.scp支持数据推送和拉取，每次都是全量拷贝，效率较低。

3.SSH验证方式

3.1 基于密码验证

知道服务器的IP端口，账号密码，即可通过ssh客户端命令登陆远程主机。

```
→ ~ ssh -p22 root@10.0.0.61
```

root@10.0.0.61 password:

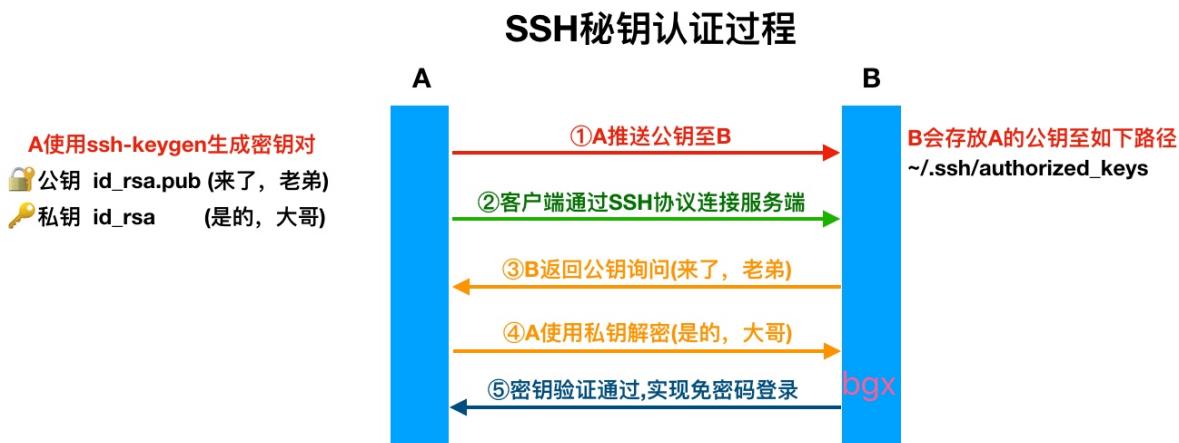
```
[root@m01 ~]#
```

#1. 密码太简单容易破解

#2. 密码太复杂容易忘记

3.2 基于密钥验证

默认情况下，通过ssh客户端命令登陆远程服务器，需要提供远程系统上的帐号与密码，但为了降低密码泄露的机率和提高登陆的方便性，建议使用密钥验证方式。



3.2.1 创建密钥

1.在A上生成非对称密钥，使用-t指定密钥类型，使用-C指定用户邮箱

```
[root@m01 ~]# ssh-keygen -t rsa -C xuliangwei@qq.com
...
#默认一路回车即可
...
#无需回车，自动应答方式生成密钥(扩展方式)
[root@m01 ~]# ssh-keygen -t rsa -C xuliangwei@qq.com -f
~/.ssh/id_rsa -P ""
```

3.2.2 推送公钥

2.将A上的公钥推送至B

```
#命令示例: ssh-copy-id [-i [identity_file]] [user@]machine
ssh-copy-id #命令
-i          #指定下发公钥的路径
[user@]      #以什么用户身份进行公钥分发（root），如果不输入，表示以当前系统用
户身份分发公钥
machine     #下发公钥至那台服务器，填写远程主机IP地址

#推送秘钥，[将A的公钥写入B的~/.ssh/authorized_keys文件中]
[root@m01 ~]# ssh-copy-id -i ~/.ssh/id_rsa.pub root@172.16.1.41

#无需交互实现秘钥推送方式
[root@m01 ~]# yum install sshpass -y
[root@m01 ~]# sshpass -p123 ssh-copy-id -i ~/.ssh/id_rsa.pub -o
StrictHostKeyChecking=no root@172.16.1.7
[root@m01 ~]# ssh -o 'StrictHostKeyChecking=no' 'root@172.16.1.7'
#登录主机
```

3.2.3 测试连接

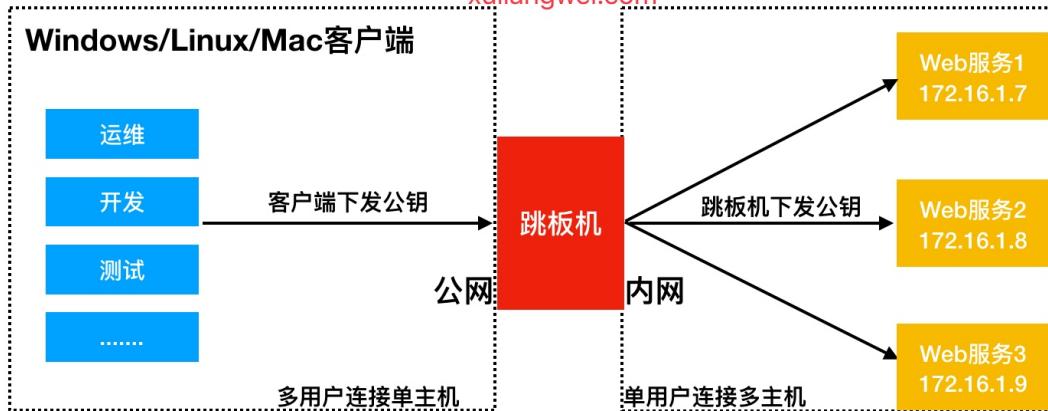
3.A通过ssh命令连接B，如果能直接连接无需密码则表示秘钥已配置成功

```
#远程登录对端主机方式
[root@m01 ~]# ssh root@172.16.1.41
[root@nfs ~]# 

#不登陆远程主机bash，但可在对端主机执行命令
[root@m01 ~]# ssh root@172.16.1.41 "hostname -i"
172.16.1.41
```

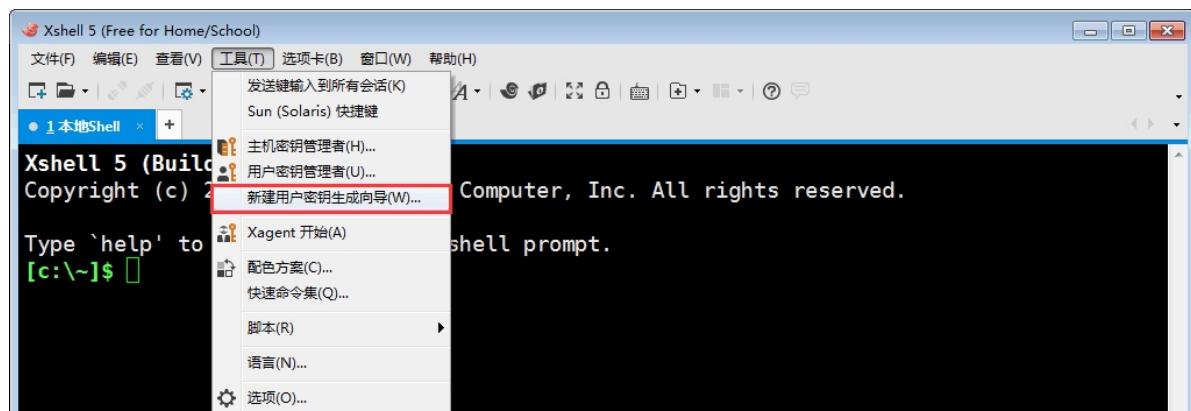
4.SSH模拟跳板机

- 用户通过Windows/MAC/Linux客户端连接跳板机免密码登录，跳板机连接后端无外网的Linux主机实现免密登录，架构图如下。
 - 实践多用户登陆一台服务器无密码
 - 实践单用户登陆多台服务器免密码



4.1 Windows下发密钥

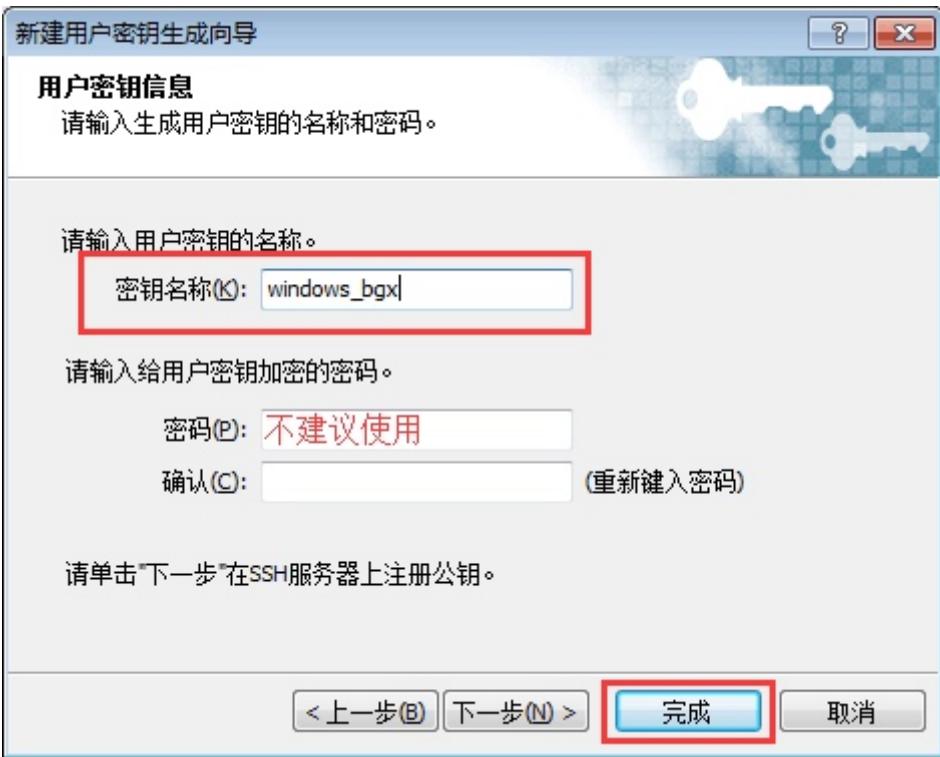
1) Xshell-->选择工具->新建密钥生成工具



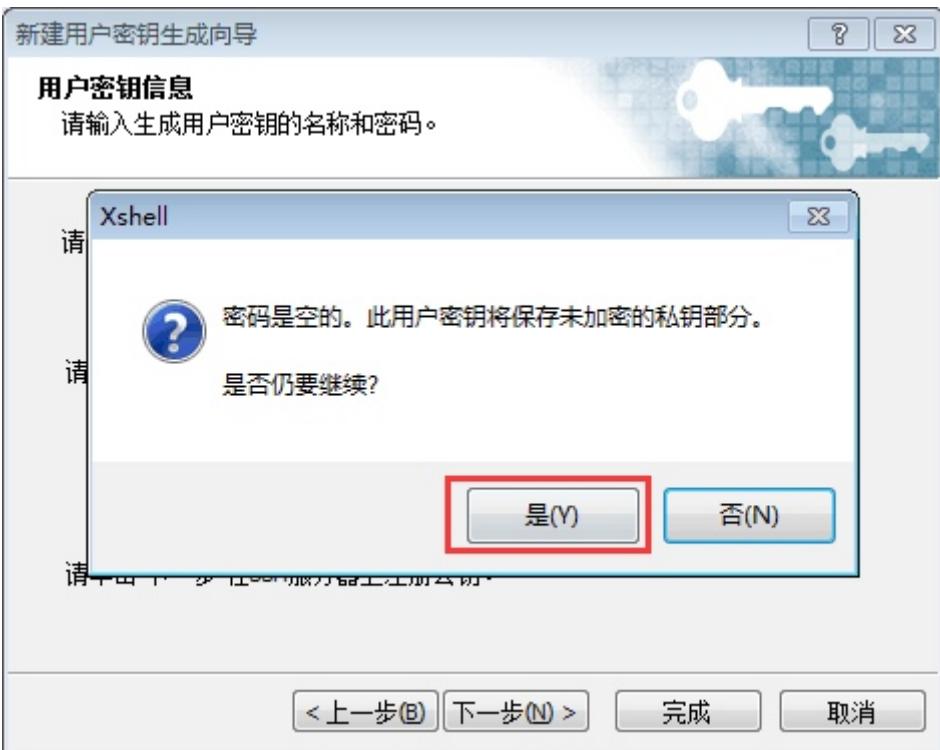
2) 生成公钥对，选择下一步



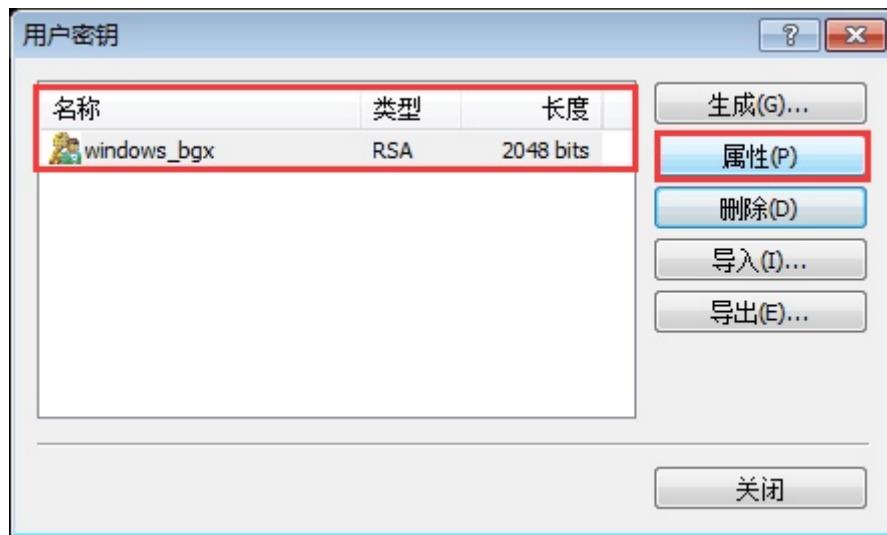
3) 填写秘钥名称。秘钥增加密码则不建议配置



4) Windows 会提示密码，继续即可



5) 生成秘钥后，点击Xshell->工具->用户秘钥管理者->选择对应秘钥的属性



6) 选择对应秘钥的公钥，将其复制



7) 将从Windows下复制好的公钥粘贴至跳板机~/.ssh/authorized_keys中，然后测试

```
[root@m01 ~]# cd ; umask 077; mkdir -p .ssh ;cd .ssh
[root@m01 .ssh]# vim authorized_keys #添加windows公钥
```

4.2 跳板机下发密钥

2. 跳板机下发公钥至后端主机

1) 在跳板机上生成秘钥对

```
[root@m01 ~]# ssh-keygen -t rsa -C Managr@qq.com
```

2) 拷贝跳板机上的密钥至后端主机，如果SSH不是使用默认22端口，使用p指定对应端口

```
[root@m01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub "-p22  
root@172.16.1.31"  
[root@m01 ~]# ssh-copy-id -i /root/.ssh/id_rsa.pub "-p22  
root@172.16.1.41"
```

3) 在m01管理机上测试是否成功登陆两台服务器

```
[root@m01 ~]# ssh root@172.16.1.41  
[root@nfs01 ~]# exit  
  
[root@m01 ~]# ssh root@172.16.1.31  
[root@backup ~]# exit
```

3.通过跳板机能实现scp拷贝文件免密码

```
[root@manager ~]# scp manager-web root@172.16.1.31:/tmp  
manager-web          100%    0     0.0KB/s   00:00  
[root@manager ~]# scp manager-web root@172.16.1.41:/tmp  
manager-web          100%    0     0.0KB/s   00:00
```

5.SSH安全优化

- SSH作为远程连接服务，通常我们需要考虑到该服务的安全，所以需要对该服务进行安全方面的配置。
 - 1.更改远程连接登陆的端口；
 - 2.禁止ROOT管理员直接登录；
 - 3.密码认证方式改为密钥认证；
 - 4.重要服务不使用公网IP地址；
 - 5.使用防火墙限制来源IP地址；
- SSH服务登录防护需进行如下配置调整，先对如下参数进行了解

```
Port 6666                      # 变更SSH服务远程连接端口  
PermitRootLogin no              # 禁止root用户直接远程登录  
PasswordAuthentication no        # 禁止使用密码直接远程登录  
UseDNS no                       # 禁止ssh进行dns反向解析，影响ssh连接效率参数  
GSSAPIAuthentication no         # 禁止GSS认证，减少连接时产生的延迟
```

将如下具体配置添加至/etc/ssh/sshd_config文件中，参数需根据实际情况进行调整

```

###SSH###
#Port 6666
#PasswordAuthentication no
#PermitRootLogin no
GSSAPIAuthentication no
UseDNS no
###END###

```

6.SSH免密Web

- 此前的ssh免密虽然可以实现跳板，但是他有很多的缺陷：
 - 1.无法知道有多少后端主机可以免密连接
 - 2.没有审计功能（无法知道用户登陆上来操作了什么）
 - 3.没有很好的权限管理机制（无法为不同的用户分配不同的主机）
- 所以：我们可以使用 `teleport web` 界面管理的跳板机来解决ssh的不足。
- 在使用 `teleport` 之前，我们需要完成此前免密的环境，因为teleport是基于此前免密的环境基础之上实现，只不过新增了权限管理、主机管理、用户管理等功能。

