西南科技大学

网络攻击与防御
实验报告

实验题目：防火墙实验

学生姓名：潘奕裴

学生学号：5120220814

一、 实验作业题目

1. 实现一个简单的防火墙
2. 无状态防火墙规则
3. 连接跟踪和状态防火墙

二、 实验思路

实现一个简单的防火墙

任务 1.1 实现一个简单的 Kernel 模块

Make 编译并构建 hello.c 文件后得到目标文件。

sudo insmod hello.ko 将 hello.ko 内核模块加载到内核中。

sudo rmmod hello 将 hello.ko 内核模块移除到内核中

```
[04/09/25]seed@VM:~/.../kernel_module$ dmesg | grep -E "Hello World!"
[04/09/25]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[04/09/25]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
[04/09/25]seed@VM:~/.../kernel_module$ sudo rmmod hello
[04/09/25]seed@VM:~/.../kernel_module$ dmesg
```

可以看到内核日志输出。

```
[ 5196.165697] Hello World!
[ 5222.298734] Bye-bye World!.
```

modinfo hello.ko 显示 hello.ko 内核模块的信息

```
[04/09/25]seed@VM:~/.../kernel_module$ modinfo hello.ko
filename:       /home/seed/Desktop/Labsetup/Files/kernel_module/hello.ko
license:        GPL
srcversion:     717A72281ACFAA8385B33A8
depends:
retpoline:      Y
name:           hello
vermagic:       5.4.0-54-generic SMP mod_unload
```

任务 1.2 使用 Netfilter 来实现一个简单的防火墙

make 编译并构建 helloFilter.c 后将内核模块加载到内核中，该模块的作用在本地生成的 IPv4 数据包即将离开本地时，会触发这两个钩子，分别输出相应的信息，尝试 ping 1.1.1.1。

```
[04/09/25]seed@VM:~/.../netfilter_hello$ sudo insmod helloFilter.ko
[04/09/25]seed@VM:~/.../netfilter_hello$ sudo dmesg -C
[04/09/25]seed@VM:~/.../netfilter_hello$ dmesg
[04/09/25]seed@VM:~/.../netfilter_hello$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=128 time=113 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=128 time=127 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=128 time=104 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 104.290/114.825/126.893/9.291 ms
[04/09/25]seed@VM:~/.../netfilter_hello$ dmesg
[ 5374.066020] *** Hello, netfilter 22222!
[ 5374.066021] *** Hello, netfilter 11111!
[ 5375.068192] *** Hello, netfilter 22222!
[ 5375.068193] *** Hello, netfilter 11111!
[ 5376.069514] *** Hello, netfilter 22222!
[ 5376.069515] *** Hello, netfilter 11111!
```

任务 1.2.1：尝试将 netfilter_hello 目录下 helloFilter.c 文件中的 hello2 函数的返回值改成 NF_DROP。

```
15 unsigned int hello2(void *priv, struct sk_buff *skb,
16                     const struct nf_hook_state *state)
17 {
18    printk(KERN_INFO "*** Hello, netfilter 22222!\n");
19    return NF_DROP;
20 }
```

hook2 低于 hook1 优先级，所以 hook2 会先被执行，因为 NF_DROP 它会将返回包丢弃，所有经 hook2 该钩子函数的数据包都会被丢弃，hook1 就不会被调用了，因此 ping 这样的网络请求就无法得到响应。但是 hook2 自身的函数体代码是会被执行，所以会输出*** Hello, netfilter 22222!

```
[04/09/25]seed@VM:~/.../netfilter_hello$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
ping: sendmsg: 不允许的操作
ping: sendmsg: 不允许的操作
ping: sendmsg: 不允许的操作
ping: sendmsg: 不允许的操作
^C
--- 1.1.1.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3074ms

[04/09/25]seed@VM:~/.../netfilter_hello$ dmesg
[ 6231.577078] *** Hello, netfilter 22222!
[ 6231.577079] *** Hello, netfilter 11111!
[ 6289.026648] The filters are being removed.
[ 6294.862106] Registering filters.
[ 6309.553714] *** Hello, netfilter 22222!
[ 6310.580027] *** Hello, netfilter 22222!
[ 6311.603827] *** Hello, netfilter 22222!
[ 6312.627816] *** Hello, netfilter 22222!
```

任务 1.2.2：用编译 packet_filter 目录下的 seedFilter.c，并添加模块到 kernel 中，并证明防火墙正在按预期工作。

seedFilter.c 需要修改，原因见问题 1。

```
hook2.hook = blockUDP;
hook2.hooknum = NF_INET_LOCAL_OUT;
hook2.pf = PF_INET;
hook2.priority = NF_IP_PRI_FIRST;
```

编译构建插入内核模块，该模块会阻止目标 IP 为 8.8.8.8 且目标端口为 53 的 UDP 数据包。（DNS 查询），测试。

```
[04/09/25]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

```
[04/09/25]seed@VM:~/.../packet_filter$ sudo dmesg  # 查看日志
[ 9107.531740] *** LOCAL_OUT
[ 9107.531741]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 9107.531747] *** Hello, netfilter 22222!
[ 9107.531887] *** Dropping 8.8.8.8 (UDP), port 53
[ 9112.526971] *** Dropping 8.8.8.8 (UDP), port 53
[ 9117.526877] *** Dropping 8.8.8.8 (UDP), port 53
```

无状态防火墙规则

任务 2.1：完成 iptables 配置后，运行 iptables -L --line-numbers

```
root@bb24a35101a5:/# iptables -A INPUT -p icmp --icmp-type echo-request -
j ACCEPT
root@bb24a35101a5:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j
 ACCEPT
root@bb24a35101a5:/# iptables -P OUTPUT DROP
root@bb24a35101a5:/# iptables -P INPUT DROP
root@bb24a35101a5:/# iptables -L --line-numbers
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     icmp --  anywhere             anywhere             icmp e
cho-request

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

Chain OUTPUT (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     icmp --  anywhere             anywhere             icmp e
cho-reply
```

Iptables 配置了：允许所有进入系统的 ICMP ping 请求和响应数据包通过，丢弃所有不符合规则的出站和入站数据包。

Chain INPUT(policy DROP)表示进入主机的数据包默认丢弃，并且设立了入站规则 1，匹配到 ping 请求包允许通过。

Chain OUTPUT(policy DROP)表示从主机发出的数据包默认丢弃，并且设立了出站规则 1，匹配到 ping 回复包允许通过。

任务 2.2：在容器 10.9.0.5 对路由器分别执行 ping 和 telnet 命令

```
[04/09/25]seed@VM:~/.../Labsetup$ docksh 8e
root@8e7b67ee0830:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.074 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.062 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.065 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.063 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.065 ms
^C
--- 10.9.0.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4082ms
rtt min/avg/max/mdev = 0.062/0.065/0.074/0.004 ms
root@8e7b67ee0830:/# telnet 10.9.0.11
Trying 10.9.0.11...
telnet: Unable to connect to remote host: Connection timed out
```

Ping 命令成功，telnet 命令失败，这是因为先前配置的防火墙规则，ping 请求可以正常通过防火墙，单 telnet 是 tcp 协议且端口为 23，防火墙入站默认策略为丢弃，所以 telnet 失败。

任务 2.3：请你修改路由器的 iptables 配置，使得容器 10.9.0.5 不可以 ping192.168.60.0/24 网络内的机器，但是 192.168.60.0/24 网络内的机器可以 ping 容器 10.9.0.5。

需要配置以下规则：阻止 10.9.0.5 向 192.168.60.0/24 发送 ICMP 请求包、允许 192.168.60.0/24 向 10.9.0.5 发送 ICMP 请求包并且允许所有 ICMP 回复包通过。

iptables -A FORWARD -s 10.9.0.5 -d 192.168.60.0/24 -p icmp --icmp-type echo-request -j DROP

iptables -A FORWARD -s 192.168.60.0/24 -d 10.9.0.5 -p icmp --icmp-type echo-request -j ACCEPT

iptables -A FORWARD -p icmp --icmp-type echo-reply -j ACCEPT

得到结果

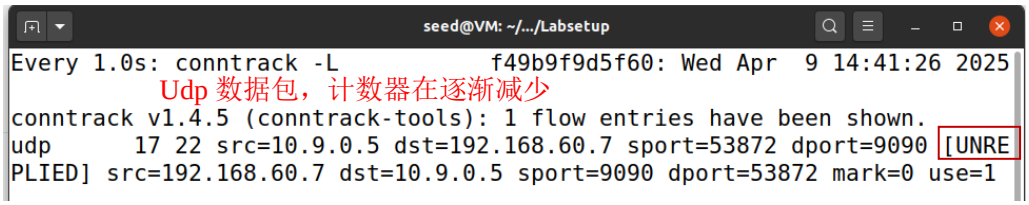
```
root@8e7b67ee0830:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
```

```
root@5c0c60f772da:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.094 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.088 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.080 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.091 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.079 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.074 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.080 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.075 ms
```

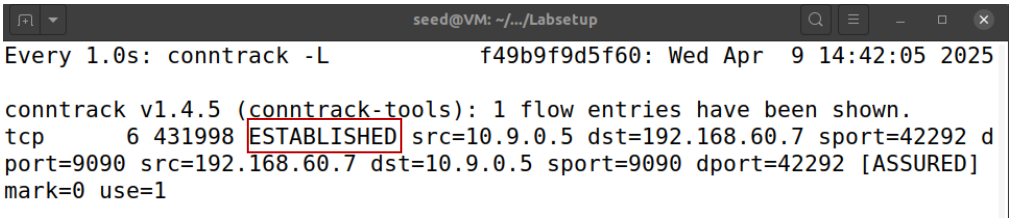
连接跟踪和状态防火墙

任务 3.1 尝试连接跟踪（Connection Tracking）

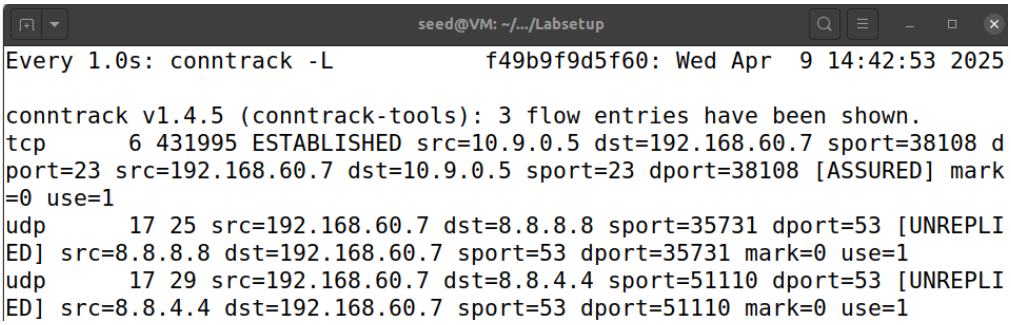
1. UDP：一条 UDP 连接记录，状态为 UNREPLIE，这是因为 UDP 不需要建立连接。计数器逐渐减少是因为连接跟踪的超时机制在运作，长时间无交互，连接记录会被删除



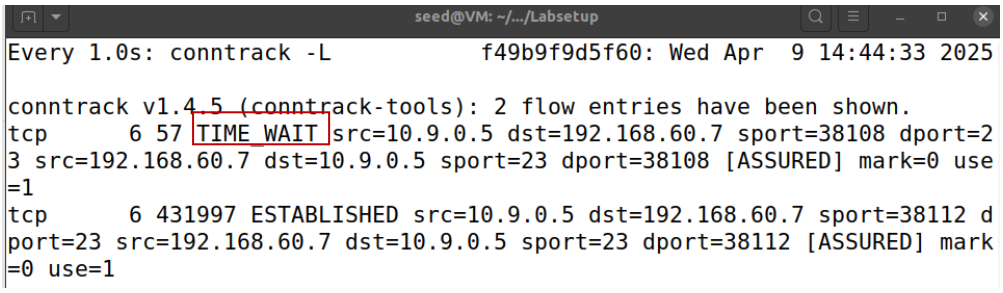
2. TCP：一条 TCP 连接记录，状态为 ESTABLISHED，这是因为 TCP 要建立连接。



3. Telnet：登陆前是存在 TCP 和 UDP 连接记录，登录后有一条 TIME_WAIT 状态的 TCP 连接，是正常状态，是 TCP 协议为保障连接可靠性和避免旧数据干扰新连接的必要机制。



登录后



任务 3.2 设置一个状态防火墙

iptables -A FORWARD -p tcp -i eth0 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT

iptables -A FORWARD -p tcp -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

iptables -P FORWARD DROP

10.9.0.5 telnet 192.168.60.7——成功

```
root@7ba312b0a3a0:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7983c062fe3e login:
```

10.9.0.5 telnet 10.9.0.1——成功

```
root@7ba312b0a3a0:/# telnet 10.9.0.1
Trying 10.9.0.1...
Connected to 10.9.0.1.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login:
```

192.168.60.7 telnet 10.9.0.5——失败

```
[04/09/25]seed@VM:~/.../Labsetup$ docksh 79
root@7983c062fe3e:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

完成任务：外网(10.9.0.0/24) 只能 telnet 192.168.60.5 ，不能 telnet 其他内网(192.168.60/24)机子，且内部主机可以访问所有内部服务器和外网的服务器

清空自定义防火墙规则后配置以下规则：允许外网（10.9.0.0/24）到 192.168.60.5 的 Telnet 新连接、允许已建立和相关的连接的流量通过、禁止外网到 192.168.60.0/24 中除 192.168.60.5 之外的 Telnet 新连接并且设置 FORWARD 链的默认策略为 DROP。

```
root@2686c826603f:/# iptables -F
root@2686c826603f:/# iptables -P FORWARD ACCEPT
root@2686c826603f:/# iptables -A FORWARD -s 10.9.0.0/24 -d 192.168.60.5 -
p tcp --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@2686c826603f:/# iptables -A FORWARD -s 10.9.0.0/24 -d 192.168.60.0/2
4 -p tcp --dport 23 --syn -m conntrack --ctstate NEW -m iprange ! --dst-r
ange 192.168.60.5 -j DROP
root@2686c826603f:/# iptables -A FORWARD -m conntrack --ctstate ESTABLISH
ED,RELATED -j ACCEPT
root@2686c826603f:/# iptables -A FORWARD -s 192.168.60.0/24 -j ACCEPT
root@2686c826603f:/# iptables -P FORWARD DROP
root@2686c826603f:/#
```

进行测试

- 外网只能 telnet 192.168.60.5，内网其他机子不行

  10.9.0.5 telnet 192.168.60.5 ——成功

```
root@7ba312b0a3a0:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c46e9659ee98 login:
```

10.9.0.5 telnet 192.168.60.7 ——失败

```
root@7ba312b0a3a0:/# telnet 192.168.60.7
Trying 192.168.60.7...
```

- 内部主机可以访问所有内部服务器和外网的服务器

192.168.60.5 telnet 192.168.60.7——成功

```
root@7983c062fe3e:/# telnet 192.168.60.7
Trying 192.168.60.7...
Connected to 192.168.60.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7983c062fe3e login:
```

192.168.60.5 telnet 10.9.0.5——成功

```
root@7983c062fe3e:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7ba312b0a3a0 login:
```

192.168.60.5 telnet 10.9.0.1——成功

```
root@7983c062fe3e:/# telnet 10.9.0.1
Trying 10.9.0.1...
Connected to 10.9.0.1.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login:
```

三、 实验分析及总结（写自己在实践过程中遇到的问题及解决方法；

本次实验**体会、收获）**

问题 1：实验 1.2.2 中不会打印出*** Dropping 8.8.8.8 (UDP), port 53

```
[04/09/25]seed@VM:~/.../packet_filter$ dmesg
[ 6687.411562] *** LOCAL_OUT
[ 6687.411564]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 6687.411571] *** Hello, netfilter 22222!
[ 6687.411723] *** LOCAL_OUT
[ 6687.411723]     61.139.2.137  --> 8.8.8.8 (UDP)
[ 6687.411725] *** Hello, netfilter 22222!
[ 6692.405308] *** LOCAL_OUT
[ 6692.405310]     61.139.2.137  --> 8.8.8.8 (UDP)
[ 6692.405316] *** Hello, netfilter 22222!
[ 6697.405356] *** LOCAL_OUT
[ 6697.405359]     61.139.2.137  --> 8.8.8.8 (UDP)
[ 6697.405364] *** Hello, netfilter 22222!
```

解决办法：添加调试

```
in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);
printk(KERN_INFO "Transefer complete.\n");

if (iph->protocol == IPPROTO_UDP) {
     udph = udp_hdr(skb);
     if (iph->daddr == ip_addr && ntohs(udph->dest) == port){
          printk(KERN_WARNING "*** Dropping %pI4 (UDP), port %
rt);
```

发现没有输出 Transefer complete,通过调试，发现是钩子触发阶段问题，blockUDP 注册在 NF_INET_POST_ROUTING 阶段，可能在某些情况下无法捕获到数据包或头部信息不可用。尝试将 blockUDP 钩子注册在更早的阶段（如 NF_INET_LOCAL_OUT），确保数据包在离开本地前被处理。修改脚本后重新编译。

```
hook2.hook = blockUDP;
hook2.hooknum = NF_INET_LOCAL_OUT;
hook2.pf = PF_INET;
hook2.priority = NF_IP_PRI_FIRST;
nf_register_net_hook(&init_net, &hook2);
```

```
[04/09/25]seed@VM:~/.../packet_filter$ sudo dmesg  # 查看日志
[ 9107.531740] *** LOCAL_OUT
[ 9107.531741]     127.0.0.1  --> 127.0.0.1 (UDP)
[ 9107.531747] *** Hello, netfilter 22222!
[ 9107.531887] *** Dropping 8.8.8.8 (UDP), port 53
[ 9112.526971] *** Dropping 8.8.8.8 (UDP), port 53
[ 9117.526877] *** Dropping 8.8.8.8 (UDP), port 53
```

问题 2：拉取容器时报错，

```
[04/09/25]seed@VM:~/.../Labsetup$ docker-compose build
HostA uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Host3 uses an image, skipping
Building Router
Step 1/2 : FROM handsonsecurity/seed-ubuntu:large
 ---> cecb04fbf1dd
Step 2/2 : RUN apt-get update     && apt-get install -y kmod      && apt-
get clean
 ---> Running in 6bd6d5aeb482
Err:1 http://archive.ubuntu.com/ubuntu focal InRelease
  Temporary failure resolving 'archive.ubuntu.com'
Err:2 http://security.ubuntu.com/ubuntu focal-security InRelease
  Temporary failure resolving 'security.ubuntu.com'
^CERROR: Aborting.
```

解决方法：修改/etc/resolv.conf 文件

```
#nameserver 10.9.0.53
#nameserver 127.0.0.53
nameserver 8.8.8.8
#search localdomain
```

没用，尝试关闭内核模块

```
[04/09/25]seed@VM:~/.../Labsetup$ sudo rmmod seedFilter
```

也没有用，重启 docker 也没用，后面重新启动虚拟机后成功拉取。

问题 3：实验 3.1，10.9.0.5 ping 192.168.60.7 时，router 使用 watch -n1 conntrack -L 观察不到变化。

```
root@8e7b67ee0830:/# ping 192.168.60.7
PING 192.168.60.7 (192.168.60.7) 56(84) bytes of data.
```

```
Every 1.0s: conntrack -L          bb24a35101a5: Wed Apr  9 14:24:09 2025

conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

解决方法：没有因为实验 2.3，设置了防火墙规则，10.9.0.5 无法 ping 到 192.168.60.7，我们重新启动容器进入即可解决，或者删除防火墙规则。

思考 1：防火墙工作于不同网络环境中，一般从哪几个方面评估防火墙性能？

吞吐量：单位时间内防火墙能够处理的最大数据包数量，反映其数据处理能力。

延迟：数据包通过防火墙所产生的时间延迟，延迟越低，性能越优。

最大并发连接数：防火墙能够同时维持的客户端与服务器端的连接总数，体现其对网络连接的承载能力。

新建连接速率：防火墙每秒能够建立的新连接数量，衡量其应对突发连接请求的能力。

转发率：防火墙能否按照网络接口的线速转发数据包，确保无阻塞地处理流量。

思考 2：从数据包流动方面简述屏蔽主机防火墙的工作原理

外部数据包先经过包过滤路由器，路由器根据预设的规则对数据包进行过滤，符合规则的数据包被允许到达内部网络中的应用网关。应用网关进一步对数据包的应用层内容进行检查和控制，若符合应用层规则，则将数据包转发至内部网络，否则拒绝。

体会及收获：

在实现 Kernel 模块与 Netfilter 防火墙时，掌握了模块编译、加载及钩子函数原理。配置 iptables 规则中，学会按需求设定策略。实验中遇钩子触发阶段、容器拉取问题，经调试、重启虚拟机解决，提升了问题排查能力。