Home server/router running Debian Squeeze

Owen T. Heisler

2013-Jun-07

Contents

About

I have set up a Soekris net5501 to act as a home router/server, using Debian squeeze. My efforts are documented here to serve as an example for others attempting similar setup scenarios. Note that this guide is only intended to cover post-installation software setup and so should be applicable to nearly any hardware.

Requirements

Here is a list of requirements:

- 100% Debian squeeze, no external software
- Stock Debian kernel
- dnsmasq server, providing DNS server (including local DNS) and DHCP server (including static IPs)
- Support dual-stack network; IPv4 and IPv6
- Provide local private network bridge
- Ethernet ports + private secured wireless
- Provide public network
- Unsecured wireless
- Simple and robust firewall, including port forwarding
- Traffic shaping to provide a better internet experience for multiple users/connections

- Carefully controlled log files

Document Conventions

- eth0 is WAN ethernet port
- aiccu is the interface of the SixXS ipv6 tunnel
- eth1-eth3 are LAN ethernet (private)
- wlan0 is private wireless
- wlan0_0 is public wireless
- br0 bridges eth1-eth3 and wlan0
- private local ipv4 network: 192.168.2.0/24
- public local ipv4 network: 192.168.3.0/24
- WAN ipv6 SixXS IP: 2001:db8::1/48

SixXS

Register with SixXS and request an tunnel and subnet. Choose your tunnel type carefully. I chose the less efficient AYIYA type because it will work behind masquerading.

Use aiccu to bring up the tunnel.

OPTIONAL: To avoid using your SixXS password in plaintext in the aiccu configuration file, add a **TIC Password** for the tunnel, then use $HANDLE/$TUNNELID as your username and the password you chose in the configuration.

aiccu.conf:

```
username $HANDLE
password $PASSWORD
protocol tic
server tic.sixxs.net
ipv6_interface aiccu
tunnel_id $TUNNELID
automatic true
requiretls false
```

Now you should have an interface named aiccu and ipv6 connectivity. Test with ping6 −c 2 www.kame.net.

Basic Networking

IPv6

Divide your /48 SixXS subnet into appropriate subnets.

First 48 bits:

1111111111111111.1111111111111111.1111111111111111.0000000000000000.0000000000000000.0000000000000000.0000000000000000

Use these 16 bits for defining subnets:

0000000000000000.0000000000000000.0000000000000000.1111111111111111.0000000000000000.0000000000000000.0000000000000000

So with /48 you would have 1 subnet.

With /50, you would have 4 subnets:

1111111111111111.1111111111111111.1111111111111111.1100000000000000.0000000000000000.0000000000000000.0000000000000000

Those 2 bits (49 and 50) determine the network. In hex, those would be:

```
0000000000000000 = 0000   xxxx:xxxx:xxxx::/50
0100000000000000 = 4000   xxxx:xxxx:xxxx:4000::/50
1000000000000000 = 8000   xxxx:xxxx:xxxx:8000::/50
1100000000000000 = c000   xxxx:xxxx:xxxx:c000::/50
```

You can calculate those with something like:

echo "obase=16;ibase=2;1000000000000000" | bc

You could further divide those 4 subnets using more bits, like perhaps 51 and 52. Here are subnets for the first xxxx:xxxx:xxxx::/50 above:

```
0000000000000000 = 0000   xxxx:xxxx:xxxx::/52
0001000000000000 = 1000   xxxx:xxxx:xxxx:1000::/52
0010000000000000 = 2000   xxxx:xxxx:xxxx:2000::/52
0011000000000000 = 3000   xxxx:xxxx:xxxx:3000::/52
```

And now another example, dividing the second primary subnet xxxx:xxxx:xxxx:c000::/50 into 8 subnets using bits 51-53:

```
1100000000000000 = c000   xxxx:xxxx:xxxx:c000::/53
1100100000000000 = c800   xxxx:xxxx:xxxx:c800::/53
1101000000000000 = c000   xxxx:xxxx:xxxx:d000::/53
1101100000000000 = c000   xxxx:xxxx:xxxx:d800::/53
1110000000000000 = c000   xxxx:xxxx:xxxx:e000::/53
1110100000000000 = c000   xxxx:xxxx:xxxx:e800::/53
1111000000000000 = c000   xxxx:xxxx:xxxx:f000::/53
1111100000000000 = c000   xxxx:xxxx:xxxx:f800::/53
```

See:

- madduck: IPv6 with Debian
- subnetcalc Debian package
- exabyte.net IPv6 Subnet Masking
- subnetonline.com IPv6 Subnet Calculator

Configuration

/etc/network/interfaces:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface (WAN)
auto eth0
#allow-hotplug eth0 # hotplugging does not seem to work reliably
iface eth0 inet dhcp

# Network bridge (LAN)
auto br0
iface br0 inet static
  hostapd /etc/hostapd/hostapd.conf  # this starts hostapd
  address 192.168.2.1
  netmask 255.255.255.0
```

```
  network 192.168.2.0
  broadcast 192.168.2.255
  bridge_ports eth1 eth2 eth3 wlan0
iface br0 inet6 static
  address (from sixxs)
  netmask 64

# Public wireless network
auto wlan0_0
iface wlan0_0 inet static
  address 192.168.3.1
  netmask 255.255.255.0
  network 192.168.3.0
  broadcast 192.168.3.255
iface wlan0_0 inet6 static
  address (from sixxs)
  netmask 64
```

Set up /etc/hosts to make local DNS work correctly. Comment this line by prepending a hash (#):

```
127.0.1.1       hostname.example.org hostname
```

And add these lines:

```
192.168.2.1     hostname.example.org hostname2.example2.org hostname
192.168.3.1     hostname.example.org hostname2.example2.org hostname
```

Test: make sure both “hostname -s” and “hostname -f” work correctly now.

dnsmasq

Edit dnsmasq.conf:

```
interface=br0
dhcp-range=private,192.168.2.51,192.168.2.250,48h

interface=wlan0_0
dhcp-range=public,192.168.3.51,192.168.3.250,48h

domain-needed
bogus-priv

# Set the NTP time server address to be the same machine as
# is running dnsmasq
dhcp-option=42,0.0.0.0

# Send microsoft-specific option to tell windows to release the DHCP lease
# when it shuts down.
dhcp-option=vendor:MSFT,2,1i

# Set the limit on DHCP leases, the default is 150
## here, raised to the maximum number of hosts on networks
dhcp-lease-max=506
```

```
# Set the DHCP server to authoritative mode. In this mode it will barge in
# and take over the lease for any client which broadcasts on the network,
# whether it has a record of the lease or not. This avoids long timeouts
# when a machine wakes up on a new network. DO NOT enable this if there's
# the slighest chance that you might end up accidentally configuring a DHCP
# server for your campus/company accidentally. The ISC server uses
# the same option, and this URL provides more information:
# http://www.isc.org/index.pl?/sw/dhcp/authoritative.php
#dhcp-authoritative
```

OPTIONAL: For static host configuration, create /etc/dnsmasq.d/static-hosts.conf:

```
# static DHCP hosts
# Optionally, use IPs from 2 to 50 (outside of DHCP range).

# example for private network
#dhcp-host=xx:xx:xx:xx:xx:xx,192.168.2.5

# example for public network
#dhcp-host=xx:xx:xx:xx:xx:xx,192.168.3.16
```

OPTIONAL: To use custom nameservers, create /etc/alt.dns:

```
# OpenNIC T2 - other servers available at opennicproject.org
nameserver   216.87.84.211                # US,CO
nameserver   2001:470:8388:10:0:100:53:20         # US,CO
nameserver   66.244.95.20                 # US,IN
nameserver   2001:470:1f10:c6::2          # US,IN
```

And uncomment line in /etc/default/dnsmasq:

```
IGNORE_RESOLVCONF=yes
```

And add line to /etc/dnsmasq.conf:

```
resolv-file=/etc/alt.dns
```

Firewall

Firewall shell scripts are slow and cumbersome. The ferm firewall utility makes writing and using firewall rules easier and much faster.

Add this line to /etc/rc.local:

```
ferm /etc/ferm/ferm.conf
```

Create /etc/ferm/ferm.conf:

```
# ferm rules
# http://ferm.foo-projects.org/

# CHAIN POLICIES
```

```
domain ip {
 table nat chain (PREROUTING OUTPUT POSTROUTING) policy ACCEPT;
}
domain (ip|ip6) {
 table raw chain (PREROUTING OUTPUT) policy ACCEPT;
 table mangle chain (PREROUTING INPUT FORWARD OUTPUT POSTROUTING) policy ACCEPT;
 table filter {
  chain (INPUT FORWARD) policy DROP;
  chain OUTPUT policy ACCEPT;
 }
}

# DNAT for inbound connection forwarding
domain ip {
 table nat {
  chain PREROUTING interface eth0 jump DNAT_MOD; # all new inbound connections
  chain DNAT_MOD { # DNAT modified in subchains of this chain
   protocol (tcp|udp) @subchain {
    jump DNAT_SKIP;
    jump DNAT_MANUAL;
    jump DNAT_UPNP;
    jump DNAT_ALLPORTS;
   }
  }
  # Keep service ports from being changed by manual/upnp/allports
  chain DNAT_SKIP {
   #protocol tcp dport 80 ACCEPT;
  }
  chain DNAT_MANUAL { # Manually forwarded ports
   # forward to LAN host
   protocol (tcp udp) dport 5155 DNAT to 192.168.2.50:5154;
   # forward to different local port
   #protocol tcp dport 81 DNAT to :80;
  }
  chain DNAT_UPNP; # UPNP forwarded ports
  chain DNAT_ALLPORTS { # Manually forwarded (ALL) ports
   DNAT to 192.168.2.5;
  }
 }
 table filter {
  chain FORWARD interface eth0 jump DNAT_FORWARD;
  chain DNAT_FORWARD {
   protocol (tcp udp) dport 5155 daddr 192.168.2.50/24 ACCEPT;
   daddr 192.168.2.5/24 ACCEPT;
   outerface (br0 wlan0_0) jump UPNP_FORWARD;
  }
  chain UPNP_FORWARD;
 }
}

# ACCEPT
# ipv6
domain ip table filter chain (INPUT OUTPUT) protocol ipv6 ACCEPT;
```

```
# lo
domain (ip ip6) table filter {
 chain INPUT interface lo ACCEPT;
 chain OUTPUT outerface lo ACCEPT;
}

# icmp
domain (ip ip6) table filter chain (INPUT OUTPUT) protocol icmp ACCEPT;

# DROP
# Host block
domain (ip ip6) table filter {
 chain (INPUT FORWARD OUTPUT) jump HOST_BLOCK;
 chain HOST_BLOCK {
  # example
  #source 77.77.77.77/24 jump HOST_BLOCK_EXEC;
 }
 chain HOST_BLOCK_EXEC {
  mod limit limit 8/min limit-burst 4 LOG log-prefix "DROP_hostblock " log-level notice;
  DROP;
 }
}

# invalid
domain (ip ip6) table filter chain (INPUT FORWARD OUTPUT) mod state state INVALID @subchain {
 mod limit limit 20/min limit-burst 2 LOG log-prefix "DROP-invalid " log-level info;
 DROP;
}

# SERVICES
# Make inbound connections resilient against DNAT manual/upnp/allports forwarding
@def &DNAT_SKIP($protocol, $port) = {
 domain ip table nat chain DNAT_SKIP protocol $protocol dport $port ACCEPT;
}

# established/related connections
domain (ip ip6) table filter chain (INPUT OUTPUT) mod state state (ESTABLISHED RELATED) ACCEPT;

# local dhcp
domain (ip ip6) table filter {
 chain INPUT protocol udp interface (br0 wlan0_0) mod multiport ports (bootps bootpc) ACCEPT;
 chain OUTPUT protocol udp outerface (br0 wlan0_0) mod multiport ports (bootps bootpc) ACCEPT;
}

# ssh
&DNAT_SKIP(tcp, ssh);
domain (ip ip6) table filter {
 chain INPUT protocol tcp dport ssh mod state state NEW @subchain {
  mod recent name SSH {
   set NOP;
   update seconds 300 hitcount 8 @subchain {
    LOG log-prefix Blocked-ssh_ log-level warning;
    DROP;
   }
```

```
  }
  ACCEPT;
 }
}

# upstream dhcp
&DNAT_SKIP(tcp, bootps:bootpc);
domain (ip ip6) table filter {
 chain INPUT protocol udp interface eth0 sport bootps:bootpc ACCEPT;
 chain OUTPUT protocol udp outerface eth0 sport bootps:bootpc ACCEPT;
}

# SixXS/AYIYA/aiccu
domain ip table filter chain OUTPUT outerface eth0 {
 protocol tcp dport 3874 ACCEPT;
 protocol udp mod multiport destination-ports (3740 5072) ACCEPT;
}

# dns
&DNAT_SKIP((tcp udp), domain);
domain (ip ip6) table filter {
 chain INPUT protocol (tcp udp) dport domain @subchain {
  interface (br0 wlan0_0) ACCEPT;
 }
 chain OUTPUT protocol (tcp udp) dport domain outerface eth0 ACCEPT;
}
domain ip6 table filter chain OUTPUT protocol (tcp udp) outerface aiccu dport domain ACCEPT;

# ntp
&DNAT_SKIP(udp, ntp);
domain (ip ip6) table filter {
 chain INPUT protocol udp dport ntp ACCEPT;
 chain OUTPUT protocol udp sport ntp ACCEPT;
}

# www/https
&DNAT_SKIP(tcp, www);
domain (ip ip6) table filter chain INPUT protocol tcp dport www ACCEPT;

# FORWARDING
# masquerading
domain ip table nat chain POSTROUTING saddr (192.168.2.0/24 192.168.3.0/24) outerface eth0 MASQUERADE;

# ipv6
domain ip table filter chain FORWARD protocol ipv6 ACCEPT;

# established connections
domain (ip ip6) table filter chain FORWARD mod state state (ESTABLISHED RELATED) ACCEPT;

# divide
domain (ip ip6) table filter chain FORWARD {
 interface eth0 {
  outerface br0 jump INET_TO_PRIVATE;
  outerface wlan0_0 jump INET_TO_PUBLIC;
```

```
 }
 interface br0 {
  outerface eth0 jump PRIVATE_TO_INET;
  outerface br0 jump PRIVATE_TO_PRIVATE;
  outerface wlan0_0 jump PRIVATE_TO_PUBLIC;
 }
 interface wlan0_0 {
  outerface eth0 jump PUBLIC_TO_INET;
  outerface br0 jump PUBLIC_TO_PRIVATE;
  outerface wlan0_0 jump PUBLIC_TO_PUBLIC;
 }
}
domain ip6 table filter chain FORWARD {
 interface aiccu {
  outerface br0 jump INET_TO_PRIVATE;
  outerface wlan0_0 jump INET_TO_PUBLIC;
 }
 interface br0 outerface aiccu jump PRIVATE_TO_INET;
 interface wlan0_0 outerface aiccu jump PUBLIC_TO_INET;
}

# internet to private
domain ip table filter chain INET_TO_PRIVATE daddr 192.168.2.5/24 ACCEPT;
domain (ip ip6) table filter chain INET_TO_PRIVATE mod limit limit 4/min LOG log-prefix "Rejected-internet-to-private "
domain ip6 table filter chain INET_TO_PRIVATE REJECT reject-with icmp6-adm-prohibited;

# internet to public
domain (ip ip6) table filter chain INET_TO_PUBLIC mod limit limit 4/min LOG log-prefix "Rejected-internet-to-public " lo
domain ip6 table filter chain INET_TO_PUBLIC REJECT reject-with icmp6-adm-prohibited;

# private to internet
domain (ip ip6) table filter chain PRIVATE_TO_INET ACCEPT;

# public to internet
domain (ip ip6) table filter chain PUBLIC_TO_INET ACCEPT;

# private to private
domain (ip ip6) table filter chain PRIVATE_TO_PRIVATE ACCEPT;

# private to public
domain ip table filter chain PRIVATE_TO_PUBLIC saddr 192.168.2.5/24 ACCEPT;
domain (ip ip6) table filter chain PRIVATE_TO_PUBLIC mod limit limit 3/min LOG log-prefix "Rejected-private-to-public "
domain ip table filter chain PRIVATE_TO_PUBLIC REJECT reject-with icmp-net-prohibited;
domain ip6 table filter chain PRIVATE_TO_PUBLIC REJECT reject-with icmp6-adm-prohibited;

# public to public
domain (ip ip6) table filter chain PUBLIC_TO_PUBLIC mod limit limit 3/min LOG log-prefix "Rejected-public-to-public " lo
domain ip table filter chain PUBLIC_TO_PUBLIC REJECT reject-with icmp-host-prohibited;
domain ip6 table filter chain PUBLIC_TO_PUBLIC REJECT reject-with icmp6-adm-prohibited;

# public to private
domain (ip ip6) table filter chain PUBLIC_TO_PRIVATE mod limit limit 3/min LOG log-prefix "Rejected-public-to-private "
domain ip table filter chain PUBLIC_TO_PRIVATE REJECT reject-with icmp-net-prohibited;
domain ip6 table filter chain PUBLIC_TO_PRIVATE REJECT reject-with icmp6-adm-prohibited;
```

```
# IGNORE
# netbios
domain (ip ip6) table filter chain INPUT protocol (tcp udp) mod multiport destination-ports (netbios-ns netbios-dgm net

# UNMATCHED
domain (ip ip6) table filter {
 chain INPUT mod limit limit 20/min LOG log-prefix "Unmatched-INPUT " log-level debug;
 chain FORWARD mod limit limit 20/min LOG log-prefix "Unmatched-FORWARD " log-level debug;
 #chain OUTPUT mod limit limit 20/min LOG log-prefix "Unmatched-OUTPUT " log-level debug;
}

# policy
domain (ip ip6) table filter chain INPUT protocol tcp mod state state NEW REJECT reject-with tcp-reset;
domain ip table filter chain INPUT mod addrtype dst-type (UNICAST MULTICAST) REJECT reject-with icmp-admin-prohibited;
domain ip6 table filter chain INPUT mod pkttype pkt-type (UNICAST MULTICAST) REJECT reject-with icmp6-adm-prohibited;
#domain ip table filter chain OUTPUT -j REJECT reject-with icmp-admin-prohibited;
#domain ip6 table filter chain OUTPUT -j REJECT reject-with icmp6-adm-prohibited;

# TCPMSS clamp
domain (ip ip6) table mangle chain FORWARD outerface eth0 protocol tcp mod tcp tcp-flags (SYN RST) SYN TCPMSS clamp-mss-
domain ip6 table mangle chain FORWARD outerface aiccu protocol tcp mod tcp tcp-flags (SYN RST) SYN TCPMSS clamp-mss-to-
```

hostapd

Start hostapd from /etc/network/interfaces rather than init.

Edit hostapd.conf:

```
interface=wlan0
bridge=br0
driver=nl80211
logger_syslog=-1
logger_stdout=-1
logger_syslog_level=2
logger_stdout_level=4
dump_file=/tmp/hostapd.dump
ctrl_interface=/var/run/hostapd

ssid=private
country_code=US
ieee80211d=1
hw_mode=g
channel=10
beacon_int=100
max_num_sta=255
rts_threshold=2347
fragm_threshold=2346
preamble=1
macaddr_acl=0
accept_mac_file=/etc/hostapd/hostapd.accept
deny_mac_file=/etc/hostapd/hostapd.deny
ignore_broadcast_ssid=0
ap_max_inactivity=300
```

```
ieee8021x=0
eap_server=0
wpa=3
wpa_passphrase=passphrase
wpa_key_mgmt=WPA-PSK WPA-PSK-SHA256
wpa_pairwise=TKIP CCMP
rsn_pairwise=CCMP
wpa_strict_rekey=1
ieee80211w=1

bss=wlan0_0
bssid=02:xx:xx:xx:xx:xx  # use your wireless MAC address, just change the first character pair
ssid=public
wpa=0
```

Traffic Shaping

I suggest using wondershaper.

Add this line to /etc/rc.local:

```
shaping.sh start
```

Create file /usr/local/sbin/shaping.sh (make it executable):

```
#!/bin/sh
set -e

case "$1" in
 start|restart)
  wondershaper eth0 $(( $(cat /var/local/down-bytes.txt)*8/1024 )) $(( $(cat /var/local/up-bytes.txt)*8/1024 ))
  ;;
 stop)
  wondershaper clear eth0
  ;;
 *)
  echo "Usage: $0 {start|restart|stop}"
  echo
  echo "start/restart: set up all shaping"
  echo "stop: remove all shaping"
  exit 1
  ;;
esac
```

Then, for bandwidth testing, create /usr/local/sbin/bandwidth-test.sh (make it executable):

```
#!/bin/sh
set -e

MAXTIME=30

# get previous limits
OLD_UP=$( cat /var/local/up-bytes.txt 2> /dev/null ) || true
OLD_DOWN=$( cat /var/local/down-bytes.txt 2> /dev/null ) || true
```

```
# disable shaping
shaping.sh stop

# start measuring
OUTPUT=$( mktemp )
FILTERED=$( mktemp )
bwm-ng -T max -I eth0 -o csv -c 0 >> $OUTPUT &
MON_PID=$!

# Use the network as much as possible, both up and down, here
 TMPFILE=$( mktemp /tmp/testfile.XXX )
 dd if=/dev/urandom of=$TMPFILE bs=1048576 count=16 2> /dev/null &
 DD_PID=$!
## DOWN
 echo "starting DOWN"
 curl --ipv4 --silent --max-time $MAXTIME "http://linux.mirrors.es.net/fedora/releases/12/Fedora/i386/iso/Fedora-12-i386
  DOWN1_PID=$!
 curl --ipv4 --silent --max-time $MAXTIME "http://nas1.itc.virginia.edu/fedora/releases/12/Fedora/i386/iso/Fedora-12-i38
  DOWN2_PID=$!
 wait $DOWN1_PID
 wait $DOWN2_PID
 echo "finished DOWN"
## UP
 sleep 2s
 wait $DD_PID
 echo "starting UP"
 curl --ipv4 --silent --max-time $MAXTIME --form file=@$TMPFILE http://www.senduit.com/ > /dev/null || true &
  UP1_PID=$!
 wait $UP1_PID
 echo "finished UP"
 rm -f $TMPFILE

# stop measuring
kill -s INT $MON_PID
wait $MON_PID
tail -n 2 < $OUTPUT | grep eth0 >> $FILTERED
MEASURED_UP=$( awk -F ";" '{ print $3 }' < $FILTERED | sed -r 's/\..*$//' )
MEASURED_DOWN=$( awk -F ";" '{ print $4 }' < $FILTERED | sed -r 's/\..*$//' )
rm -f $OUTPUT
rm -f $FILTERED

# Write new limits
echo $MEASURED_UP >| /var/local/up-bytes.txt
echo $MEASURED_DOWN >| /var/local/down-bytes.txt

# Echo new limits
MEASURED_UP_KB=$( echo "scale=1; $MEASURED_UP / 1024" | bc )
MEASURED_DOWN_KB=$( echo "scale=1; $MEASURED_DOWN / 1024" | bc )
OLD_UP_KB=$( echo "scale=1; $OLD_UP / 1024" | bc )
OLD_DOWN_KB=$( echo "scale=1; $OLD_DOWN / 1024" | bc )
echo "OLD_UP: $OLD_UP_KB"
echo "OLD_DOWN: $OLD_DOWN_KB"
echo "NEW_UP: $MEASURED_UP_KB"
echo "NEW_DOWN: $MEASURED_DOWN_KB"
```

```
# Set up shaping
shaping.sh start
```

rsyslog

Logging everything to rsyslog and then to only a few files helps keep logging simpler and disk usage more easily controlled. To see a particular facility or priority, use grep.

First, create a log rotation script /usr/local/sbin/clogrotate.sh (make it executable):

```
#!/bin/sh
# rotates channel logs; executed by rsyslog (see rsyslog.conf)
mv -f $1 ${1}.1
#gzip ${1}.1  # disable if using Flash media for root filesystem
```

Then edit rsyslog.conf:

```
# [modules]
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog   # provides kernel logging support

# [templates]
$template FullFileFormat,"%timestamp:::date-rfc3339% %syslogfacility-text%.%syslogseverity-text% %syslogtag%%msg%\n"
$ActionFileDefaultTemplate FullFileFormat

# [log file permissions]
$FileOwner root
$FileGroup adm
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022

# [channels]
$outchannel systemlog,/var/log/systemlog,104857600,clogrotate.sh /var/log/systemlog
$outchannel errlog,/var/log/errlog,26214400,clogrotate.sh /var/log/errlog

# [main]
$IncludeConfig /etc/rsyslog.d/*.conf
*.emerg                         *
*.err                           $errlog
*.notice                        $systemlog
```