



Sniffer manual

Contents

INTRODUCTION.....	3
How it works.....	3
COMMON USE CASES.....	3
Sniffing on linux host	3
Hardware port mirroring.....	4
IPTABLES mirroring.....	4
SSH traffic mirroring	4
Offline pcap files reading.....	5
INSTALLATION.....	6
Install static binary.....	6
Compile shared binary.....	6
Debian 6 squeeze.....	6
CentOS 6.2.....	7
DATABASE CONFIGURATION.....	8
MySQL	8
CONFIGURING VOIPMONITOR	9
/etc/voipmonitor.conf.....	9
TUNING VOIPMONITOR.....	13
MySQL server.....	13
Compression.....	13
innodb_buffer_pool_size = 4G (or more).....	14
innodb_flush_log_at_trx_commit = 2.....	14
Hardware.....	14
File system.....	15
UPGRADE FROM 4.2 TO 5.0.....	16
TROUBLESHOOTING.....	18

Introduction

This manual describes installation and configuration of the VoIPmonitor C++ sniffer version 5.0. If you are looking for Commercial WEB GUI manual, go to <http://www.voipmonitor.org/download> section. The upgrade procedure from versino 4.2 is described at the end of this manual.

VoIPmonitor is open source network packet sniffer for SIP and RTP VoIP protocol running on linux and with small modifications also on any posix unix. VoIPmonitor was designed to analyze quality of SIP VoIP calls based on network parameters - delay variation and packet loss according to ITU-T G.107 E-model which predicts quality on MOS scale. Calls with all relevant statistics are saved to MySQL or ODBC enabled database. Each call can be optionally saved to pcap file with either only SIP protocol or SIP/RTP/RTCP protocols. VoIPmonitor can also decode sound and play it over the commercial WEB GUI or save it to disk as WAV. Supported codecs are G.711 alaw/ulaw and commercial plugins supports G.729a/G.723/iLBC/Speex/GSM. VoIPmonitor uses jitterbuffer simulator to keep both direction of call synchronized.

How it works

VoIPmonitor is C++ program designed to handle thousands of simultaneouse calls. It listens on network interface and analyzes all SIP calls on defined SIP ports (default 5060). RTP streams which carries voice are analyzed for packet loss and variation delay (jitter). Each call is saved to MySQL or to any database supporting ODBC. SIP signalization and RTP packets can be saved to individual pcap file which can be opened with analyzers like wireshark and is also used by VoIPmonitor GUI.

Common use cases

Sniffing on linux host

You can install or compile VoIPmonitor binary directly on linux PBX or SIP server. This does not requires additional hardware and changes in network topology. The only downside is that voipmonitor consumes hardware resources - RAM, CPU and I/O workload which can affect the whole system. If it is not acceptable to share hardware for voipmonitor the second common use case is doing port mirroring.

Hardware port mirroring

Port Mirroring is used on a network switch to send a copy of network packets seen on one switch port (or an entire VLAN) to a network monitoring connection on another switch port => voipmonitor dedicated linux box. Port mirroring on a Cisco Systems switch is generally referred to as Switched Port Analyzer (SPAN); some other vendors have other names for it, such as Roving Analysis Port (RAP) on 3Com switches.

IPTABLES mirroring

IPTABLES is able to mirror traffic to another IP address. These rules are not needed in case of hardware mirroring. Rules have to be defined on the SIP server (not on the voipmonitor sniffer).

```
iptables -I PREROUTING -t mangle -i eth0 -j TEE -gateway 10.0.0.2
```

```
iptables -I POSTROUTING -t mangle -j TEE -gateway 10.0.0.2
```

This is generic rules which will mirror ALL incoming traffic from eth0 and all outgoing traffic from server to VoIPmonitor dedicated box on IP address 10.0.0.2. It is better to mirror just UDP packets

```
iptables -I PREROUTING -t mangle -i eth0 -p udp -j TEE -gateway 10.0.0.2
```

```
iptables -I POSTROUTING -t mangle -p udp -j TEE -gateway 10.0.0.2
```

SSH traffic mirroring

VoIPmonitor can read data from standard input which means that it can read traffic from any source. That source can be output from Wireshark which can run on another host piping it over SSH. Here is an example of how to do it (the command run on voipmonitor server)

```
ssh root@yourSIPserverIP "tshark -i eth0 -R 'sip or rtp' -w -" | voipmonitor -r /dev/stdin --config-file /etc/voipmonitor.conf -k
```

Offline pcap files reading

This is less common use case but also used in productions. VoIPmonitor is scripted to read pcap files which are saved on production systems to file with tcpdump or tshark or with any packet sniffer supporting pcap file format.

Installation

VoIP monitor sniffer can be installed in two ways - either as static binary which will run on any Linux distribution with kernels $\geq 2.6.18$ or compiled from sources.

Install static binary

Static binary for 32bit or 64bit can be downloaded from <http://www.voipmonitor.org/download> pages. Step by step for 64bit linux procedure:

```
tar xzf voipmonitor-*-static.tar.gz
cd voipmonitor-*-static
./install-script.sh
cp voipmonitor.conf /etc/
mysqladmin create voipmonitor
cat cdrtable.sql | mysql voipmonitor
```

Now edit configuration file `/etc/voipmonitor.conf` and run `voipmonitor`

```
/etc/init.d/voipmonitor start
```

Compile shared binary

Debian 6 squeeze

```
apt-get install build-essential subversion libmysqlclient-dev
libvorbis-dev libpcap-dev apache2 php5-mysql php5-gd mysql-server
unixodbc-dev
```

```
cd /usr/src
```

```
svn co \
https://voipmonitor.svn.sourceforge.net/svnroot/voipmonitor/tags/voip
monitor-5 voipmonitor-svn
```

```
cd voipmonitor-svn
```

```
make clean
```

```
make
```

```
make install
```

```
mkdir /var/spool/voipmonitor
```

```
mysqladmin create voipmonitor
```

```
cat cdrtable.sql | mysql voipmonitor

cp config/voipmonitor.conf /etc/

#edit file /etc/voipmonitor.conf to your needs

cp config/init.d/voipmonitor /etc/init.d/

update-rc.d voipmonitor defaults

/etc/init.d/voipmonitor start
```

CentOS 6.2

```
yum groupinstall 'Development Tools'

yum install subversion libpcap-devel mysql-devel libogg libogg-devel
vorbis-tools libvorbis libvorbis-devel mysql-server unixODBC-devel

svn co \
https://voipmonitor.svn.sourceforge.net/svnroot/voipmonitor/tags/voip
monitor-5 voipmonitor-svn

cd voipmonitor-svn

./configure

make

make install

mkdir /var/spool/voipmonitor

/etc/init.d/mysqld start

mysqladmin create voipmonitor

cat cdrtable.sql | mysql voipmonitor

cp config/voipmonitor.conf /etc/

#edit file /etc/voipmonitor.conf to your needs

cp config/init.d/voipmonitor /etc/init.d/

chkconfig --add voipmonitor

chkconfig voipmonitor on

/etc/init.d/voipmonitor start
```

Database configuration

VoIPmonitor stores CDR data to MySQL or any ODBC enabled database. Installing and configuring database is covered in previous Installation chapter. This section explains it in detail.

MySQL

This step assumes that MySQL server is started.

Create database - default is voipmonitor

```
mysqladmin create voipmonitor
```

Create SQL schema

```
cat cdrtable.sql | mysql voipmonitor
```

Running voipmonitor

Voipmonitor starts via standard init.d script `/etc/init.d/voipmonitor start|stop`. This script tells voipmonitor to load configuration file from `/etc/voipmonitor.conf`. You can run voipmonitor also from command line

```
voipmonitor --config-file /etc/voipmonitor.conf
```

which will immediately fork and run as background. If you need to see what it does, run it like this

```
voipmonitor --config-file /etc/voipmonitor.conf -k -v 1
```

Almost all configuration directives can be also provided via command line (which takes precedence over the configuration file).

To show help run voipmonitor without any parameters.

Configuring voipmonitor

/etc/voipmonitor.conf

Configuration file has only one section named [general] where all configuration directives belongs. List of directives will now follow with their description and recommendation values. Name in [] brackets is equivalent for command line which takes precedence over configuration file.

interface = eth0 [-i]

This specifies on which interface will voipmonitor listen. It can listen on one interface or on all interfaces. To listen on all interfaces use interface = any

managerport = 5029 [--manager-port <port number>]

This specifies TCP port which will voipmonitor listen for incoming connections which controls voipmonitor or for getting information about calls.

- reload configuration echo reload | nc localhost 5029

- get number of calls echo totalcalls | nc localhost 5029

- get list of calls in json format echo listcalls | nc localhost 5029

(listcalls is currently limited to max 200 calls)

managerip = 127.0.0.1

define bind address for manager interface. Default is 127.0.0.1 it is not recommended to change this unless really needed due to security. If you need it on some other IP make sure you set firewall and change the standard port for better security

sipport = 5060

define SIP ports which will voipmonitor listen. For each port make new line with sipport = port (multiple lines)

pcap-thread = yes [--pcap-thread]

Read packet from kernel in one thread and process packet in another thread. Packets are copied to non-blocking queue. Use this option if voipmonitor is dropping packets (you can see it in syslog). You can Use this option with ringbuffer.

jitterbuffer_f1 = yes
jitterbuffer_f2 = yes
jitterbuffer_adapt = yes

By default voipmonitor uses three types of jitterbuffer simulators to compute MOS score. First variant is saved into cdr.[ab]_f1 and represents MOS score for devices which has only fixed 50ms jitterbuffer. Second variant is same as first but for fixed 200ms and is saved to cdr.[ab]_f2 Third variant is adaptive jitterbuffer simulator up to 500ms Jitterbuffer simulator is the most CPU intensive task which is voipmonitor doing. If you are hitting CPU 100% turn off some of the jitterbuffer simulator.

Recommended for higher loads is to use only fixed 200ms.

ringbuffer = 20 [--ring-buffer]

This feature is the most essential parameter for high volume calls. The value is buffer size in MB allocated in kernel space. This feature will work only on kernels $\geq 2.6.32$ and libpcap ≥ 1.0 . The static version of voipmonitor contains libpcap 1.1.1. The ringbuffer is queued by packets from ethernet device and dequeued by voipmonitor. If the buffer is low and the system is overloaded (CPU or I/O) packets will be dropped. This situation will be logged to syslog.

Recommended value for high loads - 1000 (will take 1GB of RAM)

Notice: if you set this value over ~ 2000 libpcap will silently allocate nothing. Thus it is recommended to check with "ps axl|grep voipmonitor" if the voipmonitor actually uses amount of RAM you specified.

rtp-firstleg = no [--rtp-firstleg]

this is important option if voipmonitor is sniffing on SIP proxy like kamailio or openser and sees both RTP leg of CALL. In that case use this option. It will analyze RTP only for the first LEG and not each 4 RTP streams which will confuse voipmonitor. Drawback of this switch is that voipmonitor will analyze SDP only for SIP packets which have the same IP and port of the first INVITE source IP and port. It means it will not work in case where phone sends INVITE from a.b.c.d:1024 and SIP proxy replies to a.b.c.d:5060.

sip-register = no [-R]

This will save SIP REGISTER messages to PCAP files and CDR database. This is recommended only in case you really need SIP REGISTER messages. It generates millions of records in database and pcap files.

savesip = [--sip-register]

Store SIP packets to pcap file.

savertp = yes | header [-R]

Store RTP packets to pcap file or save only RTP headers and not payload (voice)

savertcp = yes [--save-rtcp]

Store RTCP packets to pcap file.

savegraph = plain [-G or --save-graph=[gzip|plain]]

This is usefull only if you have commercial WEB GUI which uses graph files for plotting graph

mos_g729 = no

enable MOS score for G.729 codec. If enabled, all cdr with 0 packet loss and stable delays will have maximum MOS of 3.92 and for loss and unstable delay MOS will be calculated according to ITU-T objective PESQ method for G.729 codec. if you want to use MOS as good search value which corellates loss and delay into single value leave it disabled (which is by default). If set to no, all calls will be calculated like it is G.711.

Recommended value = no

pcapcommand = gzip %pcap%

pcapcommand will run shell command after pcap file is closed (after call ends). %pcap% is substitution for real pcap file name. execution is guaranteed to run in serialized way (not in parallel). This example will gzip pcap file. Compressing the file to the same disk will overload disk I/O in high volume calls.

filter = udp [-f]

libpcap tcpdump style filter. Voipmonitor listens in default only for UDP packets. Unfortunatly filtering UDP packets will filter all VLAN tagged packets which means that you cannot filter only UDP if you want to listen to VLAN tagged packets.

spooldir = /var/spool/voipmonitor [-d]

This is directory where all pcap/graph/wav files are stored.

cachedir = /dev/shm/voipmonitor

store pcap and graph files to <cache/dir> and move it after call ends to spool directory. Moving all files are guaranteed to be serialized which solves slow random write I/O on magnetic or other media. Typical cache directory is /dev/shm/voipmonitor which is in RAM and grows automatically or /mnt/ssd/voipmonitor which is mounted to SSD disk.

promisc = yes [-n]

This option is only relevant if you are mirroring traffic to your network card/cards. This will not work if interface = any - in this case, use ifconfig to put your desired interfaces to promisc mode.

Default value is yes and you want to turn it off on command line use -n which will turn it off.

sqldriver = mysql

#sqldriver = odbc

#odbcdriver = mssql

#odbsdsn = voipmonitor

#odbcuser = root

#odbcpass =

voipmonitor can connect to mysql server or odbc driver. connecting voipmonitor to mssql please refer to README.mssql

mysqlhost = localhost [-h]

mysql server, default is localhost

mysqldb = voipmonitor [-b]

mysql database, default is voipmonitor

mysqltable = cdr [-t]

mysql table, default is cdr

sqlcdrtable_last30d = cdr_last30d

sqlcdrtable_last7d = cdr_last7d

sqlcdrtable_last1d = cdr_last1d

enable redundant tables which WEB GUI use for speedup searches on giant CDR tables with millions of records.

mysqlusername = root

mysql username, default is root

mysqlpassword =

mysql password, default is no password

Tuning VoIPmonitor

MySQL server

Edit /etc/mysql/my.cnf (debian/ubuntu) or /etc/my.cnf (centos) – [mysqld] section.

Compression

MySQL >= 5.1 can do table compression which greatly reduces size of the database by factor 2 without any slowdowns on CPU. Compressing CDR helps to keep whole table or its major part completely in innodb buffer (if set large) and thus minimizing time for reading database.

Enabling compression on MySQL 5.1

```
/etc/mysql/my.cnf – [mysqld] global innodb_file_per_table = 1
```

restart database

Enabling compression on MySQL > 5.1

```
MySQL> set global innodb_file_per_table = 1;  
MySQL> set global innodb_file_format = barracuda;
```

Common procedure for both MySQL versions:

```
MySQL> ALTER TABLE cdr Engine=InnoDB ROW_FORMAT=COMPRESSED  
KEY_BLOCK_SIZE=8;  
MySQL> ALTER TABLE cdr_next Engine=InnoDB ROW_FORMAT=COMPRESSED  
KEY_BLOCK_SIZE=8;  
MySQL> ALTER TABLE cdr_ua Engine=InnoDB ROW_FORMAT=COMPRESSED  
KEY_BLOCK_SIZE=8;
```

If you choose KEY_BLOCK_SIZE=2 instead of 8 the compression will be twice better but with CPU penalty on read. We have tested differences between no compression, 8kb and 2kb block size compression on 700 000 CDR with this result (on single core system – we do not know how it behaves on multi core systems). Testing query is select with group by.

```
No compression – 1.6 seconds  
8kb - 1.7 seconds  
4kb - 8 seconds
```

innodb_buffer_pool_size = 4G (or more)

This is very important variable to tune if you're using InnoDB tables. InnoDB tables are much more sensitive to buffer size compared to MyISAM. MyISAM may work kind of OK with default key_buffer_size even with large data set but it will crawl with default innodb_buffer_pool_size. Also InnoDB buffer pool caches both data and index pages so you do not need to leave space for OS cache so values up to 70-80% of memory often make sense for InnoDB only installations.

innodb_flush_log_at_trx_commit = 2

Default value of 1 will mean each update transaction commit (or each statement outside of transaction) will need to flush log to the disk which is rather expensive, especially if you do not have Battery backed up cache. Many applications are OK with value 2 which means do not flush log to the disk but only flush it to OS cache. The log is still flushed to the disk each second so you normally would not lose more than 1-2 sec worth of updates. Value 0 is a bit faster but is a bit less secure as you can lose transactions even in case MySQL Server crashes. Value 2 only cause data loss with full OS crash.

If you are importing or altering cdr table it is strongly recommended to set temporarily innodb_flush_log_at_trx_commit = 0 and turn off binlog if you are importing CDR via inserts.

Hardware

If you have not enough memory for innodb_buffer_pool_size and you are doing a lot of searches in database (through web or whatever) the most dramatic speedup is storing mysql to SSD disk. We have very good experience with Intel 5XX SSD disk which has random write 40 000 IOPS (usual SATA disk have 300-600 IOPS for random write). On that SSD disk we were not able to produce workload which is I/O bound anymore even on inserting dumped database.

File system

The fastest filesystem for voipmonitor spool directory is EXT4 with turned off journaling and other tweaks. It is better to have system/mysql on system partition and use another partition with following tweaks (the best is to have voipmonitor spool directory on dedicated disk too). Assuming your partition is /dev/sda2:

```
mke2fs -t ext4 -O ^has_journal /dev/sda2
```

```
tune2fs -O ^has_journal /dev/sda2
```

```
tune2fs -o journal_data_writeback /dev/sda2
```

```
edit /etc/fstab
```

```
/dev/sda2          /var/spool/voipmonitor  ext4
errors = remount-
ro,noatime,nodiratime,data=writeback,barrier=0 0 0
```


Upgrade from 4.2 to 5.0

Database schema has changed in version 4 in way that it needs to alter table at least two times which is so inefficient that we had to write PHP script which transforms old CDR into new structure. On SATA disk upgrading 12 millions CDR takes ~24 hours which means that the table is also locked and no CDR is possible to write during the upgrade procedure. The procedure is:

1) create new database

```
mysqladmin create voipmonitor5  
cat cdrtable.sql | mysql voipmonitor5
```

2) cd voipmonitor/scripts

edit mysql_copy_4.2to5.0.php and set appropriate constants:

```
define("HOST", "localhost");  
define("USER", "root");  
define("PASS", "");  
  
define("SOURCE_DB", "voipmonitor");  
define("DEST_DB", "voipmonitor5");
```

SOURCE_DB is the old database name, DEST_DB is the new database.

3) Run the script

```
php mysql_copy_4.2to5.0.php
```

The script can run for very long time so its recommended to run it from “screen” (apt-get install screen | yum install screen).

The speed of conversion depends a lot on two factors

- if binlog is enabled it is better to disable it in /etc/mysql/my.cn (comment out log_bin)
- in /etc/mysql/my.cnf set innodb_flush_log_at_trx_commit = 0 (and after you finish the upgrade, set it to = 2 (more secure))
- if disk is raid5 or slow SATA disk, insertion would be very slow
- if the disk with mysql is shared with /var/spool/voipmonitor and voipmonitor sniffer is running. In this case stop mysql, mv /var/lib/mysql /mnt/dedicated/; mkdir /var/lib/mysql; chown mysql

`/var/lib/mysql; mount -o bind /mnt/dedicated/mysql /var/lib/mysql –`
and start mysql. After you finish export you can move all files back
(do not forget to keep right permission on the files - user mysql).

Troubleshooting

voipmonitor does not sniff anything

- Always check if you actually see the SIP traffic. The easiest way is to run

(`apt-get install tshark` | `yum install wireshark`)

```
tshark -i eth1 -R sip
```

if you do not see traffic, make sure that the interface is UP (ip link set up dev eth1)

if you use "-i any" and you are port-mirroring traffic, make sure you put interface to promisc mode (`ifconfig eth1 promisc`; `ifconfig eth2 promisc`;) you can put this directly in `/etc/init.d/voipmonitor`

- Check `/var/log/syslog` or `/var/log/messages` for any problems related to voipmonitor. Voipmonitor logs to syslog.

- Check if voipmonitor is runningn "`ps axl |grep voipmonitor`" and is using configuration file (`--config-file /...`)