Siproxd Users Guide

Thomas Ries

Siproxd Users Guide

by Thomas Ries

Copyright © 2005-2010 Thomas Ries

This document can be freely redistributed according to the terms of the GNU General Public License.

Revision History

Revision 0.1 2005-04-10 Revised by: tries@users.sourceforge.net
Initial version
Revision 0.2 2006-07-28 Revised by: tries@users.sourceforge.net
Comment on Asterisk Scenario
Revision 0.3 2007-05-15 Revised by: tries@users.sourceforge.net
New Asterisk Config Files
Revision 0.7.1 2008-01-27 Revised by: tries@users.sourceforge.net
Plug-in API
Revision 0.8.0 2010-02-24 Revised by: tries@users.sourceforge.net
TCP, STUN plugin, config file updates

Table of Contents

README **1**
Important / Warning 1
1. Overview **2**
2. Building and Installation **3**
2.1. Prerequisites 3
2.2. Compiling and Installing 3
3. Configuration **5**
3.1. The configuration file 'siproxd.conf' 5
3.2. Command Line Options 9
4. Features **11**
4.1. Custom Firewall Module 11
4.2. Chroot() Jail 11
5. Plug-ins **12**
5.1. Plug-in API 12
5.2. Available Plug-ins 13
5.2.1. Demo Plug-in 14
5.2.2. Call Logging Plug-in 14
5.2.3. Short Dial Plug-in 14
5.2.4. Default Target Plug-in 15
5.2.5. Fix bogus Via Plug-in 15
5.2.6. STUN Plug-in 16
6. Troubleshooting **17**
6.1. Problem Reporting 17
6.2. Create a Debug Log 17
6.3. Siproxd crashes 17
7. Sample Configurations **19**
7.1. The "Standard Scenario" 19
7.2. GS BT-100 behind NAT Router running Siproxd 19
7.3. GS BT-100 with Siproxd running "in front of" a NAT router 21
7.4. Transparent SIP Proxy 22
7.5. Masquerading an Asterisk box 23

README

Important information, please read me!

Important / Warning

As it still happens that people try to mix different NAT traversal techologies together with siproxd I'll put some words here:

- Do NOT USE anything like an STUN Server together with siproxd.
- Do NOT USE any additional techologies trying to help in NAT traversal (additional firewall modules like ip_nat_sip.ko or whatever fancy stuff may tempt you).

If you do not follow the above rules, those other "helping technologies" WILL DO CONFLICT with siproxd and result in a mess.

Chapter 1. Overview

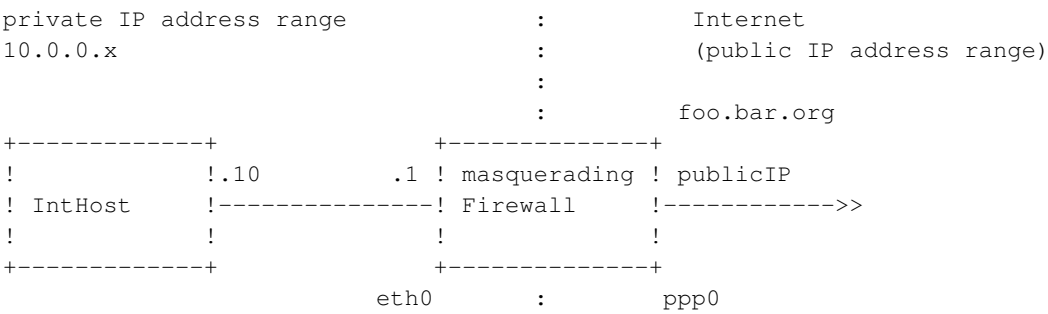
Siproxd is an proxy/masquerading daemon for the SIP protocol. It handles registrations of SIP clients on a private IP network and performs rewriting of the SIP message bodies to make SIP connections possible via an masquerading firewall. It allows SIP clients (like kphone, linphone) to work behind an IP masquerading firewall or router.

SIP (Session Initiation Protocol, RFC3261) is used by Softphones and Hardphones (Voice over IP) to initiate communication. By itself, SIP does not work via masquerading firewalls as the transfered data contains IP addresses and port numbers.

There exist so called STUN servers that allow a SIP client to figure out its public visible IP address and use this one instead. As a drawback, usually on the masquerading firewall a very wide port range must be opened up for the incoming RTP traffic. The SIP client must support STUN (which most of them do).

Siproxd uses another approach (application layer proxy) and places itself as outbound proxy in between the local SIP client and the remote client or registrar. It does rewrite the SIP traffic on the fly and also includes a RTP proxy for incoming and outgoing RTP traffic (the actual audio data). The port range to be used for receiving RTP data is configurable, so the firewall only must allow incoming traffic for a small port range.

A standard scenario would look like:



- The Firewall does IP masquerading and is running siproxd
- IntHost is running an SIP softphone (like linphone, kphone)
- The SIP address used by the softphone is sip:johndoe@foo.bar.org
- The softphone is configured to register itself at siproxd running on the firewall host (10.0.0.1) as sip:johndoe@foo.bar.org
- foo.bar.org is the domain name corresponding to the public IP address of the firewall (e.g. use some dynamic DNS service [1])

Chapter 2. Building and Installation

2.1. Prerequisites

Operating system of either:

- Linux (should work with any kernel)
- FreeBSD
- Solaris (porting is still being worked on but you may try it)

Additional required Packages:

- Libosip2 package (http://www.gnu.org/software/osip)

2.2. Compiling and Installing

It is quite simple. If you have a more-or-less standard installation and libosip2 installed at a standard location, it should be sufficient to do:

```
./configure
make
make install
```

This will install siproxd into /usr/local/. If you wish to install it into another location, specify **`--prefix=<myprefix>`** when running **`./configure`**. If you have installed libosip2 in an non-standard location use **`--with-libosip-prefix=<libosipprefix>`** to tell configure where to find libosip2 (e.g. **`--with-libosip-prefix=$HOME/lib`**).

Common features for ./configure:

```
--enable-static                 build statically linked executable
--with-libosip-prefix=DIR       use libosip2 from DIR/include and DIR/lib
--with-extra-includes=DIR       adds non standard include paths
--with-extra-libs=DIR           adds non standard library paths
```

Edit `/usr/etc/siproxd.conf` according to your situation, at least configure *`if_inbound`* and *`if_outbound`*. They must represent the interface names (e.g. on Linux: ppp0, eth1) for the inbound and outbound interfaces.

Edit `/usr/etc/siproxd_passwd.cfg` if you enable client authentication.

Start siproxd:

```
# siproxd
```

Chapter 3. Configuration

3.1. The configuration file 'siproxd.conf'

Siproxd by default searches for its configuration file in the following locations:

- `$HOME/.siproxdrc`
- `<buildingprefix>/etc/siproxd.conf`
- `/etc/siproxd.conf`
- `/usr/etc/siproxd.conf`
- `/usr/local/etc/siproxd.conf`

The following is a list of directives that do exist. Note that string values MUST NOT contain spaces or tabs. Also read the explanations included in the supplied example configuration file fro more explanation. Items with a # in front are normally disabled / not defined.

To start with siproxd in the first run, just adapt the interface definition for the inbound and outbound network interfaces (*`if_inbound`* and *`if_outbound`*).

Definition of network interfaces for the inbound network (local network where your SIP client is connected, this network normally uses IP addresses from on of the private IP ranges like 10.x.x.x, 192.168.x.x) and outbound network (your connection to the Internet, normally this interface has a public IP assigned by your provider).

```
if_inbound  = eth0
if_outbound = ppp0
```

Usually only the *`if_inbound`* and *`if_outbound`* directives will be used. The *`host_outbound`* directive comes into play when running siproxd "in front of" a NAT router. Please check the configuration examples in this document for more details. Also check the STUN plugin.

```
# host_outbound = <my_public_ip_address>
```

Access control lists for incoming SIP registrations and SIP traffic in general. These are comma separated lists of the form <IP>/<mask>, note that no spaces are allowed within the list (the configuration file parser cannot yet handle spaces).

```
# hosts_allow_reg = 192.168.1.0/24,192.168.2.0/24
# hosts_allow_sip = 123.45.0.0/16,123.46.0.0/16
# hosts_deny_sip  = 10.0.0.0/8,11.0.0.0/8
```

Port to listen for incoming SIP messages. 5060 is usually the correct choice, don't change this unless you have a reason to.

```
sip_listen_port = 5060
```

Shall siproxd run as daemon? Usually 1 is the correct choice. If you want siproxd not to daemonize and keep running in foreground and writing its output to the terminal set this to 0.

```
daemonize = 1
```

Siproxd does log using the syslog() facility when running a daemon. This setting controls how much logging is done:

- 0 - DEBUGs, INFOs, WARNINGs and ERRORs
- 1 - INFOs, WARNINGs and ERRORs
- 2 - WARNINGs and ERRORs
- 3 - only ERRORs
- 4 - absolutely nothing

```
silence_log = 0
```

If siproxd is started as root, it can drop the root privileges and change its user ID at startup. It also can put itself into a chroot() jail (see 4.2 for details)

```
user = nobody
# chrootjail = /var/lib/siproxd/
```

Where to store the current registrations and the cycle in seconds to perform the cyclic writing. This allows siproxd to remember registration across a restart. An empty value means we do not save registrations. The specified directory path must exist.

```
registration_file = /var/lib/siproxd/siproxd_registrations
autosave_registrations = 300
```

Where to create the PID file.

```
pid_file = /var/run/siproxd/siproxd.pid
```

Enable/disable the RTP proxy. This must always be enabled. In some future release this directive will become obsolete and will be removed.

```
rtp_proxy_enable = 1
```

Port range (UDP) that siproxd will use for incoming and outgoing RTP traffic. A firewall must be configured to allow traffic from and to these ports (UDP only). By default the range 7070 up to (and including) 7089 is used. This allows up to 10 simultaneous calls (2 ports per call). If you need more simultaneous calls, increase the range.

```
rtp_port_low  = 7070
rtp_port_high = 7089
```

Timeout for an RTP stream. If for the specified number of seconds no data is relayed on an active stream, it is considered dead and will be killed.

```
rtp_timeout = 300
```

DSCP (differentiated services) value to be assigned to RTP and SIP UDP packets. Allows QOS aware routers to handle different types traffic with different priorities.

```
rtp_dscp = 46
sip_dscp = 0
```

If a REGISTER request does not contain an `Expires` header or `expires=` parameter in the `Contact` header, this number of seconds will be used and reported back to the UA in the answer.

```
default_expires = 600
```

TCP inactivity timeout: For TCP SIP signalling, this indicates the inactivity timeout (seconds) after that an idling TCP connection is disconnected. Note that making this too short may cause multiple parallell registrations for the same phone. This timeout must be set larger than the used registration interval.

```
tcp_timeout = 600
```

Timeout for connection attempts in msec: How many msecs shall siproxd wait for an successful connect when establishing an outgoing SIP signalling connection. This should be kept as short as possible as waiting for an TCP connection to establish is a BLOCKING operation - while waiting for a connect to succeed not SIP messages are processed (RTP is not affected).

```
tcp_connect_timeout = 500
```

TCP keepalive period: For TCP SIP signalling, if > 0 empty SIP packets will be sent every 'n' seconds to keep the connection alive. Default is off.

```
tcp_keepalive = 20
```

If siproxd is used as registration server and authentication is wanted, define the following directive. If *`proxy_auth_realm`* is defined (a string), clients will be forced to authenticate themselfs to the proxy (for registration only). To disable Authentication, simply comment out this line. Default is disabled.

```
# proxy_auth_realm = Authentication_Realm
```

The password to be used for authentication may be a global one

```
# proxy_auth_passwd = some_password
```

or on a per user base, stored in its own file. *`proxy_auth_pwfile`* takes precedence over *`proxy_auth_passwd`*

```
# proxy_auth_pwfile = /etc/mysiproxd_passwd.cfg
```

To enable additional debug output of siproxd. This is a bit pattern representing the following items. Default is 0x0 - disabled. See below in this document for information on how to create a debug log file.

- DBCLASS_BABBLE 0x00000001 // babble (like entering/leaving fnc)
- DBCLASS_NET 0x00000002 // network
- DBCLASS_SIP 0x00000004 // SIP manipulations
- DBCLASS_REG 0x00000008 // Client registration
- DBCLASS_NOSPEC 0x00000010 // non specified class
- DBCLASS_PROXY 0x00000020 // proxy
- DBCLASS_DNS 0x00000040 // DNS stuff
- DBCLASS_NETTRAF 0x00000080 // network traffic
- DBCLASS_CONFIG 0x00000100 // configuration
- DBCLASS_RTP 0x00000200 // RTP proxy
- DBCLASS_ACCESS 0x00000400 // Access list evaluation
- DBCLASS_AUTH 0x00000800 // Authentication

```
debug_level = 0x00000000
```

You may connect to this port from a remote machine and receive the debug output. This allows bettwer creation of debug output on embedded systems that do not have enough memory for large disk files. Port number 0 means this feature is disabled.

```
debug_port = 0
```

Some UAs (SIP clients) will always use the host/ip they register TO as host part in the registration record (which will be the inbound ip address/hostname of the proxy) and can not be told to register a different host (public IP address). This Mask feature allows to force such a UA to be masqueraded to a different host. Siemens SIP Phones seem to need this feature. Normally disabled.

```
# mask_host=local.ip.of.sipphone
# masked_host=public.domaind.org
```

User Agent Masquerading: Siproxd can masquerade the User Agent string of your local UAs. Useful for Providers that do not work with some specific UAs (e.g. sipcall.ch - it does not work if your outgoing SIP traffic contains an Asterisk UA string...) Default is to do no replacement.

```
ua_string = Siproxd-UA
```

Use ;rport in via header: may be required in some cases where you have a NAT router that remaps the source port 5060 to something different and the registrar sends back the responses to port 5060. Default is disabled (0)

```
use_rport = 0
```

Siproxd itself can be told to send all traffic to another outbound proxy. You can use this feature to 'chain' multiple siproxd proxies if you have several masquerading firewalls to cross. Normally disabled.

```
# outbound_proxy_host = my.outboundproxy.org
# outbound_proxy_port = 5060
```

Outbound proxies can be specified on a per-domain base. This allows to use an outbound proxy needed for ProviderA and none (or another) for ProviderB. Multiple domain specific proxies may be specified, each one with one set of the following directives. Note: These directives must always be specified as a triple, skipping one of them will affect later definitions.

```
#outbound_domain_name = freenet.de
#outbound_domain_host = proxy.for.domain.freende.de
#outbound_domain_port = 5060
```

Siproxd supports dynamic loadable plug-ins. Such plug-ins are loaded during runtime and do not require recompilation of the executable. This allows the easy addition of specific functionality to siproxd. Even 3rd party functional extensions are possible without the requirement to patch and rebuild the siproxd source code with each new release.

Note: Dynamic loading of shared libraries is not supported on all platforms. If a platform does not support it, plug-ins can still be used but they will be statically linked during the build process of siproxd. The configuration ("loading" the plugins) is identical. For more information on this topic you may familiarize yourself with libltdl.

Note: As the plug-in mechanism uses LTDL, the plugins to load MUST use a .la extension and not an .so extension! Trying to load an plugin using xxx.so as it's name will fail.

```
# plugin_dir: MUST be terminated with '/'
plugindir=/usr/lib/siproxd/

# List of plugins to load:
#load_plugin=plugin_demo.la
load_plugin=plugin_logcall.la
```

Each plugin does manage it's own set of configuration options. They are named like plugin_<pluginname>_xxxxx. For the detailed description of configuration settings, refer to the plugin description.

```
plugin_demo_string = This_is_a_string_passed_to_the_demo_plugin
```

3.2. Command Line Options

Siproxd knows the following command line options:

```
-h, --help                      help
-d, --debug <pattern>           set debug-pattern
-c, --config <cfgfile>          use the specified config file
```

```
-p, --pid-file <pidfile>        create pid file at <pidfile>
```

These options take precedence over the values configured in the configuration file.

Chapter 4. Features

4.1. Custom Firewall Module

The API

make your library

example code

```
./configure --with-custom-fwmodule=LIBRARY.a
```

4.2. Chroot() Jail

Create chroot jail

What files must be present? To be completed!

Chapter 5. Plug-ins

5.1. Plug-in API

Siproxd plug-ins are dynamic loadable libraries and must provide 3 functions towards siproxd. As we make use of some libltdl features we do some internal macor magic - the PLUGIN_xxx funcation names are actually CPP macros that will expand in unique names. Th have this working properly the PLUGIN_NAME must be #defined before the plugins.h header file is included:

```
#define PLUGIN_NAME     plugin_myplugin
#include "plugins.h"
[...]
int  PLUGIN_INIT(plugin_def_t *plugin_def);
int  PLUGIN_PROCESS(int stage, sip_ticket_t *ticket);
int  PLUGIN_END(plugin_def_t *plugin_def);
```

The `PLUGIN_INIT` function is called when the plug-in is loaded during startup of siproxd. The plug-in must define the following 4 fields of the plugin_def structure:

1. `api_version`
2. `name`
3. `desc`
4. `exe_mask`

Example code fragment:

```
/* API version number of siproxd that this plug-in is built against.
 * This constant will change whenever changes to the API are made
 * that require adaptions in the plug-in. */
plugin_def->api_version=SIPROXD_API_VERSION;

/* Name and descriptive text of the plug-in. Those item MUST NOT be
   on the stack but either allocated via malloc (and then freed
   of course) or a static string in the plug-in. */
plugin_def->name=strdup("plugin_demo");
plugin_def->desc=strdup("This is just a demo plugin without any purpose");

/* Execution mask - during what stages of SIP processing shall
 * the plug-in be called. */
plugin_def->exe_mask=PLUGIN_DETERMINE_TARGET|PLUGIN_PRE_PROXY;
```

The `PLUGIN_PROCESS` function is called at the requested SIP processing stages (see 'execution mask'). Your processing will be done here.

The `PLUGIN_END` function is called at shutdown of siproxd and gives the plug-in the opportunity to clean up and properly shutdown itself.

Note: The previously allocated 'name' and 'desc' must be freed by the plug-in. If you did use a static string then of course you must not try to free() anything.

Minimum required clean up procedure:

```
int  PLUGIN_END(plugin_def_t *plugin_def){
   /* free my allocated rescources (if allocated via malloc only) */
   if (plugin_def->name) {free(plugin_def->name); plugin_def->name=NULL;}
   if (plugin_def->desc) {free(plugin_def->desc); plugin_def->desc=NULL;}
   return STS_SUCCESS;
}
```

For a simple example refer to the simple demonstration plug-in `plugin_demo`.

Each plug-in can have its own set of configuration parameters in `siproxd.conf`. The plug-in has to define a `cfgopts_t` structure and call `read_config` during its initialization. Look at `plugin_demo` for a simple example. The naming of the config parameters is by definition `plugin_name_`option.

5.2. Available Plug-ins

The following plug-ins are provided with siproxd:

1. `plugin_demo`

   Demo plug-in. Provides the basic framework to be used for plug-ins.

2. `plugin_logcall`

   Very simplistic call logging to syslog.

3. `plugin_shortdial`

   Quick Dial feature.

4. `plugin_defaulttarget`

   Incoming calls to a non-existing UA are redirected to a specific target (catch-all).

5. `plugin_fix_bogus_via`

   Fixes broken VIA headers on incoming calls.

6. `plugin_stun`

   Periodically checks with an STUN server for the public IP address of siproxd.

Some of the plug-ins are described in more detail in the following chapters.

5.2.1. Demo Plug-in

Name: plugin_demo

Purpose: To be used as skeletton for your own plug-ins.

Configuration options:

```
plugin_demo_string = This_is_a_string_passed_to_the_demo_plugin
```

Description:

This plug-in can be used as framework for your own plug-ins. It contains the required code for the API and also shows how to load plug-in specific configuration parameters.

5.2.2. Call Logging Plug-in

Name: plugin_logcall

Purpose: Does a very simplistic call logging to syslog.

Configuration options:

```
none
```

Description:

The numbering starts with "1" ("*01") and every following "plugin_shortdial_entry" entry will allocate the following position. It is not possible to freely assign the positions.

5.2.3. Short Dial Plug-in

Name: plugin_shortdial.c

Purpose: Provides a quick-Dial feature.

Configuration options:

```
# The first character is the "key", the following characters give
# the length of the number string. E.g. "*00" allows speed dials
# from *01 to *99. (the number "*100" will be passed through unprocessed)
plugin_shortdial_akey = *00
#
# *01 sipphone echo test
plugin_shortdial_entry = 17474743246
# *02 sipphone welcome message
plugin_shortdial_entry = 17474745000
```

Description:

Allows the definition of quick dial entries. E.g. *01 to *99 can be defined to redirect the caller. Note: Currently only the user part (phone number) can be replaced, the domain part will not be changed (means a short dial tarket of sip:111@other.domain.com will not work). The '*' character can be chosen freely (plugin_shortdial_akey). Note: To call a real number like "*12" you have to dial "**12"

5.2.4. Default Target Plug-in

Name: plugin_defaulttarget

Purpose: Incoming calls to non-existing local UAs are redirected to another SIP URI.

Configuration options:

```
# Log redirects to syslog
plugin_defaulttarget_log = 1
# target must be a full SIP URI with the syntax
# sip:user@host[:port]
plugin_defaulttarget_target = sip:defaulttarget@some.sip.domain:port
```

Description:

Incoming SIP calls directed to a non-existing (registered) local UA will be redirected to another target. This basically implements a catch-all feature. The new target can be any SIP URI and is not required to be local.

5.2.5. Fix bogus Via Plug-in

Name: plugin_fix_bogus_via

Purpose: Fixes broken VIA headers on incoming SIP requests.

Configuration options:

```
# Incoming (from public network) SIP messages are checked for broken
# SIP Via headers. If the IP address in the latest Via Header is
# part of the list below, it will be replaced by the IP where the
# SIP message has been received from.
plugin_fix_bogus_via_networks = 10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
```

Description:

Fixes broken VIA headers on incoming SIP requests (inspired by Ralph Babel, see http://babel.de/art20080317a.html for more info). Can be applied if you have remote UAs calling you from the Internet and those UAs do have crappy Via headers (like private IPs because there is some NAT involved on their side).

5.2.6. STUN Plug-in

Name: plugin_stun

Purpose: Uses an external STUN server to determine the public IP address of the host running siproxd.

Configuration options:

```
# Plugin_stun
#
# Uses an external STUN server to determine the public IP
# address of siproxd. Useful for "in-front-of-NAT-router"
# scenarios.
plugin_stun_server = stun.ekiga.net
plugin_stun_port = 3478
# period in seconds to request IP info from STUN server
plugin_stun_period = 300
```

Description:

Does contact the configured STUN server at startup and then cyclically at the configured interval to determine the public visible IP address of the host running siproxd. Useful for setups that have changing public IP addresses and siproxd is running in the "in-front-of-NAT-router" scenario.

Chapter 6. Troubleshooting

6.1. Problem Reporting

If you encounter problems/crashes and ask for support, please include as much information as possible. Very helpful is a debug log that has been recorded at the time of the misbehavior. Also include the exact versions of the siproxd package and libosip2 that you are using. You should also include your `siproxd.conf`.

6.2. Create a Debug Log

The easiest way to generate a debug log is:

1. make sure siproxd is not started as daemon ('daemonize = 0' in the config file)
2. start siproxd: **`$ ./siproxd -d -1 2>debug.log`**
3. reproduce the error
4. include the generated `debug.log` in your error report

Another possibility of to use TCP logging. This method is recommended if you run siproxd on a router with limited disk space (e.g. an embedded system). To enable TCP logging:

1. Edit the configuration file and set *`debug_port`* to 5050 (or any other TCP port number you like).
2. Restart siproxd
3. **`$ telnet <IP_of_siproxd> 5050 > debug.log`**

You may prefer to use netcat instead of telnet. Note: The TCP debug port is bound to all available interfaces on the system, make sure no unauthorized people (like from the outbound network) can connect.

6.3. Siproxd crashes

If siproxd crashes, a stack back trace usually is helpful to me:

1. start siproxd in the debugger (daemonize set to 0):

```
$ gdb ./src/siproxd
```

```
(gdb) set args -c /path/to/siproxd.conf
```

```
(gdb) run
```

2. reproduce the crash

3. use gdb to print the stack backtrace:

```
(gdb) info thread
...
(gdb) bt
#0  0x400ec9ee in __select ()
#1  0xbffff6f8 in ?? ()
#2  0x804a5c2 in main (argc=3, argv=0xbffffc54) at siproxd.c:186
#3  0x4005bcb3 in __libc_start_main (main=0x804a30c <main>, argc=3,
    argv=0xbffffc54, init=0x8049a08 <_init>, fini=0x804edac <_fini>,
    rtld_fini=0x4000a350 <_dl_fini>, stack_end=0xbffffc4c)
    at ../sysdeps/generic/libc-start.c:78
(gdb)
```

4. copy-paste all the output and include it in your problem report.

Chapter 7. Sample Configurations

Check also the FAQ in the siproxd package.

7.1. The "Standard Scenario"

Scenario:

```
private IP address range             :          Internet
10.0.0.x                             :          (public IP address range)
                                     :
                                     :         foo.bar.org
+-------------+               +--------------+
!             !.10         .1 ! masquerading ! publicIP
! IntHost     !---------------! Firewall     !------------>>
!             !               !              !
+-------------+               +--------------+
                          eth0       :        ppp0
```

The Firewall does IP masquerading and is running siproxd. IntHost is running an SIP softphone (like linphone, kphone). The SIP address used by the softphone is `sip:johndoe@foo.bar.org`. The softphone is configured to register itself at siproxd running on the firewall host (10.0.0.1) as `sip:johndoe@foo.bar.org`. `Foo.bar.org` is the domain name corresponding to the public IP address of the firewall (e.g. use some dynamic DNS service like DynDNS).

Firewall configuration (iptables):

```
# allow incoming SIP and RTP traffic
iptables -A INPUT -m udp -p udp -i ppp0 --dport 5060      -j ACCEPT
iptables -A INPUT -m udp -p udp -i ppp0 --dport 7070:7089 -j ACCEPT
```

Firewall configuration (ipchains):

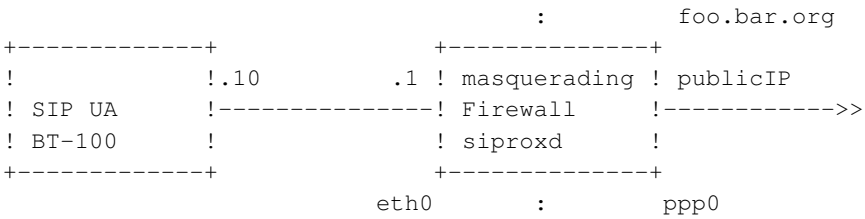
```
# allow incoming SIP and RTP traffic
ipchains -A input --proto udp --dport 5060      -j ACCEPT
ipchains -A input --proto udp --dport 7070:7089 -j ACCEPT
```

The first line will allow incoming SIP traffic. The second line will allow incoming RTP traffic on the ports 7070 - 7089 (the default port range used by siproxd for incoming RTP traffic).

7.2. GS BT-100 behind NAT Router running Siproxd

Scenario:

```
private IP address range             :          Internet
10.0.0.x                             :          (public IP address range)
                                     :
```



Siproxd is running on the same host as the masquerading firewall. The SIP phone is a Grandstream BudgeTone-100. In this example the external SIP registrar used is sipphone.com (http://www.sipphone.com/).

siproxd.conf:

```
if_inbound  = eth0
if_outbound = ppp0
hosts_allow_reg = 10.0.0.0/24
sip_listen_port = 5060
daemonize = 1
silence_log = 1
user = siproxd
registration_file = /var/lib/siproxd_registrations
pid_file = /var/run/siproxd/siproxd.pid
rtp_proxy_enable = 1
rtp_port_low  = 7070
rtp_port_high = 7089
rtp_timeout = 300
default_expires = 600
debug_level = 0
debug_port = 0
```

Firewall configuration (iptables):

```
# allow incoming SIP and RTP traffic
iptables -A INPUT -m udp -p udp -i ppp0 --dport 5060      -j ACCEPT
iptables -A INPUT -m udp -p udp -i ppp0 --dport 7070:7089 -j ACCEPT
```

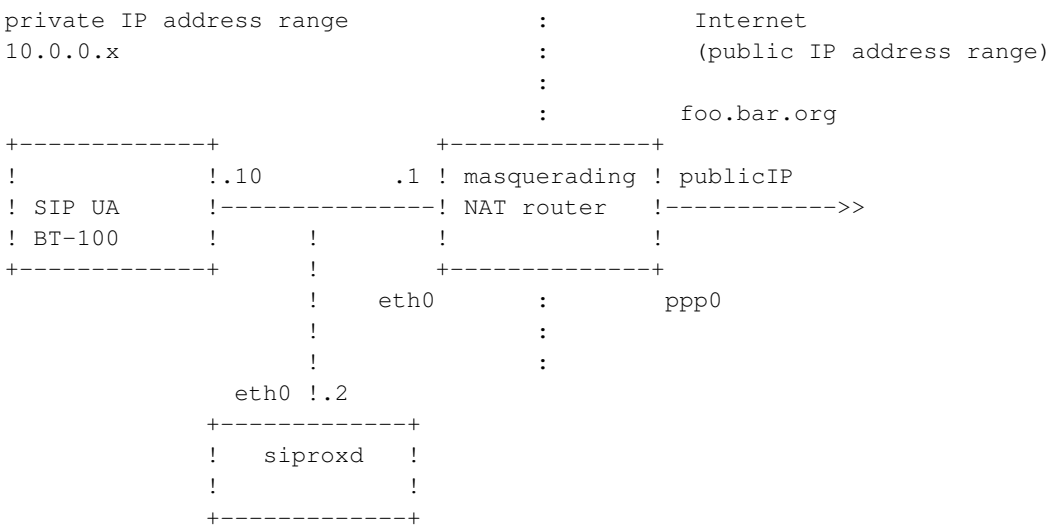
Phone configuration (only the relevant items are listed):

```
IP Address:             10.0.0.10
Subnet Mask:            255.255.255.0
Default Router: 10.0.0.1
DNS Server 1:           <DNS Server of your Internet provider>
SIP Server:             proxy01.sipphone.com
Outbound Proxy: 10.0.0.1
SIP User ID:            1747669xxxx
Authenticate ID:        1747660xxxx
Authenticate Passwd:    *********
Name:                   Your Name Here
Use DNS SRV:            no
User ID is phone #:     no
Sip Registration:       yes
Unregister on reboot:no
Register expiration:    60
```

```
Early Dial:             no
local SIP port: 5060
local RTP port: 5004
Use random port:        yes
NAT traversal:  no
Use NAT IP:             <empty>
Subscribe for MWI:      No
Send DTMF:              via RTP (RFC2833)
```

7.3. GS BT-100 with Siproxd running "in front of" a NAT router

Scenario:



Siproxd is running on 10.0.0.2. The masquerading NAT router (e.g. a ADSL NAT router that cannot run any user applications).

siproxd.conf:

```
if_inbound  = eth0
if_outbound = eth0
host_outbound = foo.bar.org
hosts_allow_reg = 10.0.0.0/24
sip_listen_port = 5060
daemonize = 1
silence_log = 1
user = siproxd
registration_file = /var/lib/siproxd_registrations
pid_file = /var/run/siproxd/siproxd.pid
rtp_proxy_enable = 1
rtp_port_low  = 7070
rtp_port_high = 7089
rtp_timeout = 300
```

```
default_expires = 600
debug_level = 0
debug_port = 0
```

NAT router configuration:

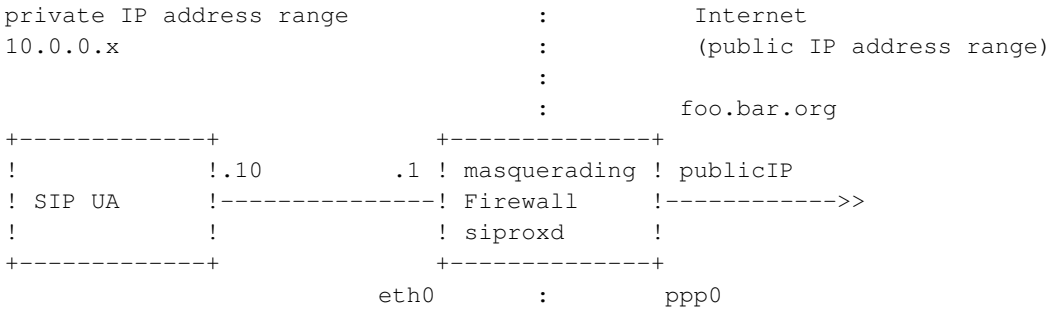
```
forward all incoming traffic on 5060/udp to 10.0.0.2
forward all incoming traffic from 7070/udp - 7089/udp to 10.0.0.2
```

Phone configuration:

```
IP Address:             10.0.0.10
Subnet Mask:            255.255.255.0
Default Router: 10.0.0.1
DNS Server 1:           <DNS Server of your Internet provider>
SIP Server:             proxy01.sipphone.com
Outbound Proxy: 10.0.0.2
SIP User ID:            1747669xxxx
Authenticate ID:        1747660xxxx
Authenticate Passwd:    *********
Name:                   Your Name Here
Use DNS SRV:            no
User ID is phone #:     no
Sip Registration:       yes
Unregister on reboot:no
Register expiration:    60
Early Dial:             no
local SIP port: 5060
local RTP port: 5004
Use random port:        yes
NAT traversal:  no
Use NAT IP:             <empty>
Subscribe for MWI:      No
Send DTMF:              via RTP (RFC2833)
```

7.4. Transparent SIP Proxy

Scenario:



You may have a SIP UA (Phone) that does not allow the specification of an outbound proxy. If siproxd is running on the masquerading router, the following configuration will do so called

transparent proxying. The firewall will redirect outgoing SIP messages to siproxd, however the local Client is not aware of it.

siproxd.conf:

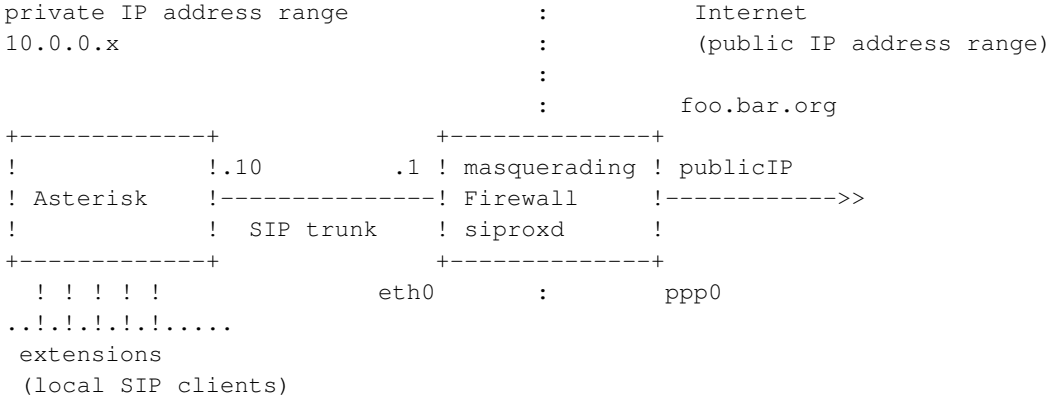
```
if_inbound  = eth0
if_outbound = ppp0
hosts_allow_reg = 10.0.0.0/24
sip_listen_port = 5060
daemonize = 1
silence_log = 1
user = siproxd
registration_file = /var/lib/siproxd_registrations
pid_file = /var/run/siproxd/siproxd.pid
rtp_proxy_enable = 1
rtp_port_low  = 7010
rtp_port_high = 7019
rtp_timeout = 300
default_expires = 600
debug_level = 0
debug_port = 0
```

Firewall configuration (iptables):

```
# redirect outgoing SIP traffic to siproxd (myself)
iptables -t nat -A PREROUTING -m udp -p udp -i eth0 \
                              --destination-port 5060     -j REDIRECT
# allow incoming SIP and RTP traffic
iptables -A INPUT -m udp -p udp -i ppp0 --dport 5060      -j ACCEPT
iptables -A INPUT -m udp -p udp -i ppp0 --dport 7070:7089 -j ACCEPT
```

7.5. Masquerading an Asterisk box

Scenario:



Siproxd can also be used to masquerade an Asterisk server. The Asterisk server will register itself as a SIP UA (Client) to an external SIP registrar. In this example this would be again sipphone.com. As

Asterisk does not allow to specify an SIP outbound proxy we use the same setup for transparent proxying. The context values of the asterisk configuration probably must be adapted to fit your needs.

siproxd.conf:

```
if_inbound  = eth0
if_outbound = ppp0
hosts_allow_reg = 10.0.0.0/24
sip_listen_port = 5060
daemonize = 1
silence_log = 1
user = siproxd
registration_file = /var/lib/siproxd_registrations
pid_file = /var/run/siproxd/siproxd.pid
rtp_proxy_enable = 1
rtp_port_low  = 7070
rtp_port_high = 7089
rtp_timeout = 300
default_expires = 600
debug_level = 0
debug_port = 0
```

Firewall configuration (iptables):

```
# redirect outgoing SIP traffic to siproxd (myself)
iptables -t nat -A PREROUTING -m udp -p udp -i eth0 \
                     --source 10.0.0.11 --destination-port 5060 -j REDIRECT
# allow incoming SIP and RTP traffic
iptables -A INPUT -m udp -p udp -i ppp0 --dport 5060      -j ACCEPT
iptables -A INPUT -m udp -p udp -i ppp0 --dport 7070:7080 -j ACCEPT
```

Asterisk configuration (SIP related part):

Note: Very important are the fromuser and fromdomain keywords in the client section. They are required to have Asterisk send the correct From headers in SIP dialogs. The used Asterisk version is 'SVN-branch-1.4-r62331M'.

With newer Asterisk versions, it is no longer required to have a separate REGISTER definition, this can be made implicit in the SIP trunk config.

```
; sip.conf:

[general]
port = 5060           ; Port to bind to (SIP is 5060)
bindaddr = 0.0.0.0    ; Address to bind to (all addresses on machine)
context = from-sip-external ; Send unknown SIP callers to this context

useragent = PBX       ; NOTE: some providers (e.g sipcall.ch) do simply
                      ;       not work with the default "AsteriskPBX"
                      ;       UA String.

; Network Settings
nat=never
```

```
localnet = 10.0.0.0/24
domain = 10.0.0.10

; Codecs
disallow=all
allow=gsm       ; 13 Kbps
allow=ulaw      ; 64 Kbps
allow=alaw      ; 64 Kbps
autoframing = yes

; SIP Settings
canreinvite = no      ; important!

; the following are just my settings I use, however
; I dont' consider them critical
allowexternaldomains = yes
allowexternalinvites = yes
allowguest = yes
allowsubscribe = no
allowtransfer = yes
alwaysauthreject = no
autodomain = yes
callevents = no
compactheaders = no
dumphistory = no
g726nonstandard = no
ignoreregexpire = no
jbenable = no
jbforce = no
jblog = no
maxcallbitrate = 384
maxexpiry = 3600
minexpiry = 180
notifyringing = no
pedantic = no
promiscredir = no
recordhistory = no
relaxdtmf = no
rtcachefriends = no
rtsavesysname = no
rtupdate = no
sendrpid = yes
sipdebug = no
t1min = 100
progressinband = no
;register =
t38pt_udptl = no
trustrpid = no
usereqphone = no
videosupport = no
```

The Trunk definition looks like:

```
; users.conf:

[general]
;
```

```
; Full name of a user
;
fullname = New User
userbase = 200
;
; Create voicemail mailbox and use use macro-stdexten
;
hasvoicemail = yes
;
; Set voicemail mailbox 6000 password to 1234
;
vmsecret = 1234
;
; Create SIP Peer
;
hassip = yes
hasiax = no
;
; Create H.323 friend
;
;hash323 = yes
;
; Create manager entry
;
hasmanager = no
;
; Remaining options are not specific to users.conf entries but are general.
;
callwaiting = yes
threewaycalling = yes
callwaitingcallerid = yes
transfer = yes
canpark = yes
cancallforward = yes
callreturn = yes
callgroup = 1
pickupgroup = 1
host = dynamic
localextenlength = 3
allow_aliasextns = no
allow_an_extns = no
hasagent = no
hasdirectory = no


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Local SIP UAs
; = locally connected phones. nothing special here.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
[201]
callwaiting = yes
cid_number = 201
context = local_sip
email = e@mail
fullname = Full Name
group =
hasagent = yes
```

```
hasdirectory = yes
hasiax = no
hasmanager = no
hassip = yes
hasvoicemail = yes
host = dynamic
mailbox = 201
secret = sip_password
threewaycalling = yes
zapchan =
registeriax = no
registersip = yes
vmsecret = 1234


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; SIP Trunks
; these are masqueraded via siproxd
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
[trunk_1]
disallow = all
allow = gsm,ulaw,alaw,adpcm,speex,g729,g723
callerid =
contact = 17476691234       ; IMPORTANT
context = DID_trunk_1
dialformat = ${EXTEN:1}
fromdomain = proxy01.sipphone.com
fromuser = 17476691234      ; IMPORTANT
group =
hasexten = no
hasiax = no
hassip = yes
host = proxy01.sipphone.com
insecure = very
port = 5060
provider =
registeriax = no
registersip = yes
secret = sip_password
trunkname = Custom - sipphone1234
trunkstyle = customvoip
username = 17476691234
```