Dante FAQ

Inferno Nettverk A/S
Oslo Research Park
Gaustadalléen 21
NO-0349 Oslo
Norway

Date: 2006/01/08 18:33:48

See http://www.inet.no/dante/FAQ/ for the latest version of this faq.
Send updates/corrections to the public mailinglist dante-misc@inet.no.

1 Reporting problems or bugs in Dante

1.1 Why was I told to read this in response to my superb problem/bug report?

Because you were unable to provide a usable report. Since there are so many of you, we prefer to refer you to this FAQ rather than repeating ourselves over and over.

1.2 Why is "it does not work" not a good report?

It's a common misconception among users who have never done any programming that by stating something along the lines of "it does not work", or "I get a segmentation fault" (these are real examples, believe it or not), the programmers at Inferno Nettverk will without a shred of doubt be able to pin-point that one line, that one line among the thousands of other lines, responsible for it "not working" or "segmentation faulting".

The fact is, any one of the tens of thousands of lines could be the "bad" line, we don't have time to re-read them all each time we get a "it does not work" message.

1.3 How can I provide you with the information you need?

First of all, look in the config file you are using. If you are using the Dante client, it's /etc/socks.conf. If you are using the Dante server, it's /etc/sockd.conf. Both files contain by default the lines "logoutput:". Set "logoutput:" to some file that Dante can log output to.

If the problem is with the client, also uncomment the line containing "debug". If the problem is with the server, also add "-d" to whatever arguments you usually start the server with.

Repeat whatever you did to produce the error and look at the log messages. Do they provide any hints?

If it looks like a user error, you configured something wrong, etc. you can try sending a mail to dante-misc@inet.no, a public mailing list where someone might be kind enough to help you if you formulate your question in a understandable way. And remember, unless you are a customer of Inferno Nettverk, no one owes you any support or help, you are the one asking someone to spend their time to help you, not the other way around.

If it looks like a bug, first perform the following step:

1. Make sure that you are using the latest version of Dante, even if the latest version is a "prerelease" and you are using official release. Always check ftp.inet.no:/pub/socks/ to see if there's a later version before sending a bugreport.

   Next, depending on whether the suspected bug is in the client or sever, do one of the following:

   If it's a bug in the client:

   (a) Compile the program you are trying to socksify with the compiler option "-g".
   (b) Compile Dante with "-g" (–enable-debugging option to configure).
   (c) Set debug and logoutput as described above.
   (d) cp /dev/null <logoutput file> (to make sure it's empty.)
   (e) ktrace/truss/strace/<whatever> socksify <bad program>. Save the output to a file. If you have a program that can trace library calls too (late version of Solaris truss and redhat Linux apparently do), please include information about that too.
   (f) If a coredump was generated do

   ```
   $ gdb <bad program> <coredump>
   (gdb) backtrace 50
   ```

   Save the output to a file.

   If it's a bug in the server:

   (a) Compile Dante with "-g" (–enable-debugging option to configure).
   (b) Set debug and logoutput as described above.
   (c) cp /dev/null <logoutput file> (to make sure it's empty.)
   (d) If the server crashed/exited due to the bug, a coredump should be generated. If you do not find a coredump in the directory you started the Dante server from you need to find out why. Some possible reasons are: a) your shell has disabled coredumps (check what ulimit(1))

says. b) some systems by default disable coredumps on processes that are either setuid, or that are run by root.
When you have found the coredump do

```
$ gdb <Dante server> <coredump>
(gdb) backtrace 50
```

Save the output to a file.

Then:

2. In addition to the server/client specific files mentioned below, include the output from configure and the file "include/autoconf.h".

3. Collect the output from configure and the file include/autoconf.h, the <logoutput file>, gdb/ktrace/truss/strace/whatever output in a orderly fashion so it's easy to read and understand, and send it to us. Make sure you include information about what all the IP addresses in the logfiles are (10.0.0.1 is "client using lynx", 10.0.0.2 is "remote webserver", etc.).

   The address to use for Dante bugs is dante-bugs@inet.no. See the BUGS file coming with Dante.

4. If you have the possibility to create a temporary account for one of the Dante developers in the case we are unable to reproduce the error on our own systems, please mention that in the bugreport (but don't create the account until we ask for it, we rarely need it).

2 System installation

2.1 How do I configure the client?

2.1.1 Setting up the configuration file

The configuration file is usually called */etc/socks.conf* and needs information about where the server can be found. To configure it follow these easy steps:

1. Read the manual page (*socks.conf*). It can also be found online.

2. Read the manual page again.

3. Determine the IP address and port of the socks server you wish to use.

4. If the server uses anything but socks version 5 you should probably also determine this.

Lets assume that server is at address is 10.1.1.1, port 1080 and that it is a normal socks server (such as Dante) which supports socks version 5.

You would then enter the following in */etc/socks.conf*:

```
route {
       from: 0.0.0.0/0   to: 0.0.0.0/0   via: 10.1.1.1 port = 1080
       proxyprotocol: socks_v5
}
```

The Dante socks library should now be able to connect to the socks server.

2.1.2 It doesn't work

If it doesn't work you might wish to add the lines below to the beginning of the file. When you try again you should get more information about what is happening, which will hopefully help you determine what the problem is:

```
debug: 1
logoutput: stdout
```

2.1.3 I want to do something more complicated

Reading the manual page is a good start. Then have a look at the example configuration-files found in the archive (and via ftp).

2.2 How do I set up the server?

2.2.1 Creating a simple server configuration file

Setting up the server is not significantly more difficult than setting up the client. The server usually uses the filename */etc/sockd.conf*. To configure the server follow these easy steps:

1. Read the manual pages (*sockd.conf* and *sockd*). They can also be found online.

2. Read the manual pages again.

3. Find out the name or IP address of the interface on the internal side of the network.

4. Find out the name or IP address of the interface(s) on the external side of the network.

5. Determine which networks (address/netmask pair) you wish to allow to connect to the server.

6. Determine if you wish to use or need any special authentication to use the server. If not, the server will most likely not need to be started as root.

7. Create a user (e.g. *sockd*) which the server can run as when it doesn't need to perform privileged operations. Using *nobody* is not recommended.

8. Determine what users will be allowed to do via the server (i.e. which ports should they be allowed to use).

Armed with this knowledge you can now start writing the configuration file. It generally consists of tree parts:

- General configuration; setting external interfaces etc.
- Client access rules; who is allowed to connect to the server
- Client usage rules; what users can do when they have connected

Let's start with the general rules:

```
internal: fxp0 port = 1080
external: fxp1
method: username none
#user.privileged: root
user.notprivileged: sockd
logoutput: stderr
```

This sets the internal and external interfaces and which users should be used by the server. It allows users to connect with username authentication, but allows access without authentication. Running as root might not be necessary if you do not use any authentication methods.

Then we add the client access rules. These are placed after the first lines.

```
client pass {
from: 10.1.1.0/24 to: 0.0.0.0/0
log: connect disconnect
}
```

These lines allow clients on the 10.1.1.0 network to connect to any external address. The server logs connects and disconnects.

Finally we add the access rules. This example does not place any limits upon usage of the server.

```
#allow bind to ports greater than 1023
pass {
from: 0.0.0.0/0 to: 0.0.0.0/0 port gt 1023
command: bind
log: connect disconnect
}

#allow outgoing connections (tcp and udp)
pass {
from: 0.0.0.0/0 to: 0.0.0.0/0
command: connect udpassociate
log: connect disconnect
}

#allow replies to bind, and incoming udp packets
pass {
       from: 0.0.0.0/0 to: 0.0.0.0/0
       command: bindreply udpreply
       log: connect error
}

#log the rest
block {
       from: 0.0.0.0/0 to: 0.0.0.0/0
       log: connect error
}
```

These lines allow all clients (which have been allowed to connect) to use the socks *bind*, *connect* and *udpassociate* operations. This gives users full access through the socks server; clients can bind ports on the server (and receive replies), connect with TCP through the server, and send (and receive) UDP packets.

The *bind* command is typically used by *ftp* clients (using active ftp), and the first pass command allows clients to bind a port (above 1023) on the server.

The second pass command allows outgoing traffic; *connect* operations (TCP) and transmission of UDP packets.

The third *pass* command controls *replies*; this might be the reply to a bind request or it may be the reply to a UDP-packet, since UDP is packetbased. Note that nothing is done to verify that it's a *genuine* reply, that is in general not possible anyway.

The *block* command at the end ensures that operations that do not match any of the above are logged. In the above configuration there will likely not be many blocked operations since the commands are very liberal in what they allow. The same result could have been achieved much simpler by a single *pass* command. The structure above has been used to make it simpler to make modifications. By changing the networks matched in the *from* and *to* commands above it is possible to limit operations to specific subnets, only allow outgoing connections to some addresses etc.

The complete sockd configuration file should now look like this:

```
## general configuration

internal: fxp0 port = 1080
external: fxp1
method: username none
#user.privileged: root
user.notprivileged: sockd
logoutput: stderr

## client access rules

client pass {
from: 10.1.1.0/24 to: 0.0.0.0/0
log: connect disconnect
}


## server operation access rules

#allow bind to ports greater than 1023
pass {
from: 0.0.0.0/0 to: 0.0.0.0/0 port gt 1023
command: bind
log: connect disconnect
}

#allow outgoing connections (tcp and udp)
```

```
pass {
from: 0.0.0.0/0 to: 0.0.0.0/0
command: connect udpassociate
log: connect disconnect
}

#allow replies to bind, and incoming udp packets
pass {
       from: 0.0.0.0/0 to: 0.0.0.0/0
       command: bindreply udpreply
       log: connect error
}

#log the rest
block {
       from: 0.0.0.0/0 to: 0.0.0.0/0
       log: connect error
}
```

This was a simple example to get you started. As you probably noticed when you read the manual page the server can do a lot more.

Remember to test your configuration before you put it into production usage. When you are sure that the server works you might wish to redirect the logoutput to a file (by changing the *logoutput* command).

2.3 Where can I find configuration file examples?

Several examples are included in the archive. They can also be downloaded via ftp.

2.4 SuSE 7

1. Install the Dante package from SuSE (Version 1.1.2 is included in SuSE 7)

2. Edit the /etc/rc.config and set the line START_SOCKD="yes"

3. Edit the config file /etc/sockd.conf and reboot Linux (or start sockd -D if you know what you are doing). Be sure that you have one client pass rule which allows you client to connect from your internal network and one pass rule that allows a client that is connected to use the proxy to the desired destinations (this is most likely to everywhere = 0.0.0.0/0)

Thanks to: Andreas John <dante@systemdesign.net>

3 How can I socksify the whole system?

On some systems doing the global equivalent of running a program with socksify is possible. You should however not do this unless you know what you are doing.

3.1 Linux

In Linux you can easily do the following:

```
edit the file /etc/ld.so.preload
and add two lines saying:
   libdl.so
   libdsocks.so
```

Thanks to: Dag Wieers <dag@wieers.com>.

Dag Wieers also has a (RedHat) SPEC file which contains scripts for doing this automatically. It can be found at: ftp://dag.wieers.com/home-made/dante/dante.spec

3.2 Generic

There is another way that gives you more control over what will and what will not be socksified. Instead of using the wrapper-script 'socksify', you could add

```
LD_PRELOAD="libdl.so libdsocks.so"
export LD_PRELOAD
```

to the environment. (/etc/profile /.profile /.bashrc)
Thanks to: Dag Wieers <dag@wieers.com>.

3.2.1 Why do I want to be able to control this?

In same rare cases programs fail to work in a socksified environment (e.g. acroread 4.0 for Linux segfaults) or programs like rpm depend on a clean build-environment to get its (library) dependencies.
Thanks to Dag Wieers <dag@wieers.com>.

3.2.2 How can I easily unsocksify the current shell?

By issuing

```
unset LD_PRELOAD
```

Thanks to: Dag Wieers <dag@wieers.com>.

4 Technical Problems

4.1 Why doesn't the socksify program work with glibc-2?

If glibc is not correctly installed (old header files are still installed) socksify won't work if logging is enabled. The server will also crash when it exits. Upgrade the header files (/usr/include) and compile again.

4.2 Why doesn't socksify work on cvs?

It does work. A problem however is that cvs usually ends up calling a setuid program (rsh, ssh, whatever). See section 4.3.

4.3 Why doesn't socksify work on setuid programs like ssh, rsh, etc?

The reason is that these programs are setuid. Most systems will ignore the work socksify has done before running these programs and the result becomes the the same as not using socksify on them.

Solution: strip off the suid bit if possible or recompile the binary (rsh, ssh, etc) with explicit socks support (adding "-ldsocks" to the linkoptions is usually enough.)

4.4 Why doesn't socksify work on HP-UX 11.00?

You might have to install patch *PHSS_22478*, or one of its successors, which adds *LD_PRELOAD* support to the loader.

Thanks to: albert chin (china@thewrittenword.com)

4.5 Why doesn't socksify work on AIX?

AIX does not support *LD_PRELOAD*, which is required for socksify to work. The alternative is static compilation, as specified in the `ftp://ftp.inet.no/pub/socks/extracted/doc/README.usage` file.

4.6 Why do I get undefined references to __bswap_32 during compilation on Linux?

This is likely due to a bug in one of the system header files. A workaround has been added in version 1.1.14.

4.7 How can I more easily get the Dante server to work when the external interface is using dynamic IP addresses?

Either specify the external address as a interfacename (available since Dante v1.1.7), in which case the Dante server will automatically use the new address whenever it changes, or if you must use a IP address, try the below.

Automatically running a script that changes the "external:" line in your sockd.conf and then sends the server a SIGHUP when you get a new address should be simple enough though.

A example of such a script is the following, modify to suit.

```
#!/bin/sh -
#
# usage: script.sh <new external ipaddress>

: {SOCKD_CONF='/etc/sockd.conf'}

if [ ! -w ${SOCKD_CONF} ]; then
   echo "${SOCKD_CONF} does not exist or is not writable"
   exit 1
fi
```

```
ex - +'1,$s/^external:.*/external: '$1'/ | wq' ${SOCKD_CONF}
kill -HUP `cat /var/run/sockd.pid`
```

As you see, this is intended to be called as "<scriptname> <external ip>", where <external ip> is your external ip. We assume you can easily get the external ip from your ppp script. *Thanks to: Tin Yu Mak <makty@usa.net>.*

4.8 I have problems rotating Dante's logfiles. What can I do except log via syslog?

Dante now reopens the logfiles on SIGHUP, so you should not have any problems, just rename the file and send the Dante server a SIGHUP signal.

A quick and dirty solution for dealing with most daemons that don't like having their logs rotated. Copy them instead of moving the files and then redirect an echo into them. Kick it off via cron every day at 11:59. I get 7 days of rotated logs like this:

```
sleep 60
DAY=`date +%a`
cp /var/log/banned.log /var/log/banned.log.${DAY}
echo > /var/log/banned.log
```

Thanks to: James Coe <starascendant@hotmail.com>.

Another solution:

I use dante under supervise with multilog for log rotation (based on size, not date). This will run Dante under supervise (and limit its memory usage to 5 megabytes) and redirect its logging to /var/log/dante/current, rotated based on its size.

http://cr.yp.to/daemontools.htmlhttp://cr.yp.to/daemontools.html for more information.

```
   /service/dante/run:

#!/bin/sh

exec 2>&1
exec softlimit -m 5000000 -p 30 -c 0 \
        /usr/sbin/sockd -f /etc/sockd.conf -N 1 -l # -d

/service/dante/log/run:

#!/bin/sh

exec setuidgid logging multilog t /var/log/dante
```

Thanks to: Michael T. Babcock" <mbabcock@fibrespeed.net>

5 Authentication

5.1 How do I setup a password file for authenticating users?

The Dante server uses the standard system passwordfile. On many systems, the Dante "user.privileged" id must be set to root or somebody with special access to read the encrypted passwordfile.

Put the server in a chroot jail to manage a separate passwordfile.

5.2 How do I use LDAP authentication with Dante?

First, make sure that you compile Dante with support for PAM. The configure script should do this for you if the PAM development libraries are installed. Under RedHat Linux, make sure you have the PAM-devel RPM installed.

You will also have to add LDAP support to your PAM implementation, if it is not already there. Check if you have /lib/security/pam_ldap.so or the equivalent. If not, use pam_ldap from http://www.padl.com/.

pam_ldap gets its configuration from /etc/ldap.conf. Make sure you set host to your LDAP server, and base to your base DN.You can also set pam_filter and pam_group_dn to restrict logins to particular objecttypes or members of particular groups.

Point sockd at LDAP by creating /etc/pam.d/sockd. An exapmle of this files contents is;

```
#%PAM-1.0
auth       sufficient /lib/security/pam_ldap.so
auth       required   /lib/security/pam_unix_auth.so try_first_pass
account    sufficient /lib/security/pam_ldap.so
account    required   /lib/security/pam_unix_acct.so
password   required   /lib/security/pam_ldap.so
```

Finally, you need to configure sockd.conf to restrict access based on PAM. this is accomplished with the following:

```
# allowable authentication methods for socks-rules.
# values are username, none, rfc931 and pam
method: pam none

# allowable authentication methods for client-rules.
# values are username, none, rfc931 and pam
clientmethod: pam none

# Client rules, controls who may connect
#
## everyone who authenticates is allowed to use tcp
## and udp
pass {
       from: 0.0.0.0/0 to: 0.0.0.0/0
       protocol: tcp udp
```

```
        log: connect disconnect error
        method: pam
}

# last line, block everyone else.  This is the default but if you
provide
# one  yourself you can specify your own logging/actions
block {
        from: 0.0.0.0/0 to: 0.0.0.0/0
        log: connect error
}
```

Thanks to: Sean Burford <sean.burford@adelaide.edu.au>.

5.2.1 ... and against Microsoft Active Directory?

To authenticate against a Microsoft Active Directory the following bare minimum example configuration for 'test.org' with an active directory server at 123.123.123.123 of /etc/ldap.conf Directory is what you would require:

```
host 123.123.123.123
base cn=Users,dc=test,dc=org
rootbinddn cn=Administrator,cn=Users,dc=test,dc=org
pam_filter objectclass=user
pam_login_attribute cn
```

With the password for the Administrator user in the file /etc/ldap.secret and chowned to 0600.

Thanks to: Alexander Cohen <A.Cohen@latrobe.edu.au>.

5.3 Technical details

5.3.1 Why do I always get an initial success to my erroneous username/password with Dante?

If username/password authentication is used, the socks client will always get a successful response at first. The username/password is not checked until the full socks request has been received from the client, at which point the request will fail if the username/password is incorrect in the same way it would if there was no "pass" rule for the request.

The reason for this is that due to the generality of the Dante socks-rules, we do not know what authentication we will be using for this particular clients request, until we have read the request. That means, the same client address could in theory use one type of authentication for connections to one address, and another authentication type for connections to another address.

As of version 1.1.16, this has been changed a little. If you are using only one type of username/password-based authentication scheme, the username/password will be checked at an point early enough to tell the client the username/password was wrong.

6 For system administrators

6.1 What happens if I need to restart the Dante server?

If you kill one sockd process (any one), the rest of the processes will cope as best as they can.

This means that if you kill the main sockd process, clients that have already established their socks-session will continue to run. These sockd processes will be orphaned (their parent pid will be init), and will not be able to accept new clients.

You can at this point however start the Dante server again. The newly started Dante server will then accept clients as usual, but will have no relation to the processes created by the previous Dante server. Those processes will continue to run by themselves until all their clients have finished, at which point they will die my themselves.

7 For programmers

7.1 What does a socksified application need to care about?

A socksified application will normaly work without any extra steps needing to be done, within the limits of the socks standard. The exception to this is the following:

select(2)/poll(2) applications combining non-blocking sockets with select(2)/poll(2) need to make sure they can handle the select(2)/poll(2) call being interrupted. This is due to how the Dante clientlibrary implements support for non-blocking connect(2).