Inferno Nettverk A/S

Dante Technical Specification

Data Traffic Log Format

Dante version 1.2.3

March 21, 2011

Contents

1 Introduction

This document describes the format used to log traffic in the Dante SOCKS proxy server. The information that is logged includes IP-addresses and port numbers for clients that connect to the Dante server, the addresses used by the Dante server itself and the machines that the Dante server is used to facilitate communication with. Log data can be written to *syslog*, a file, *stdout*, or *stderr*.

The Dante server can additionally log connection content, status information, errors and debugging data. This document does not describe these type of log data.

2 Example

A typical usage scenario is using the Dante server to initiate a connection to a remote host. The line below shows an example of this, corresponding to a line logged for a SOCKS CONNECT request, made by a client at the address 127.0.0.1, to open a TCP connection to port 7 on a remote host with the IP-address 10.0.0.2.

```
pass(1): tcp/connect [: 127.0.0.1.1044 127.0.0.1.2044 ->
                        10.0.0.1.1983 10.0.0.2.7
```

Note that the log entry has been broken over two lines, this is only for readability; in the log file the entry is only a single line.

Four sets of numbers occur in the message, corresponding to the IP-address and port numbers of the two pairs of TCP connection endpoints involved in the command; the first pair for the TCP connection between the client and the Dante SOCKS server, and the second for the connection between the Dante SOCKS server and the remote host.

Replacing the IP-addresses and port numbers with symbolic representations makes the position of each set of values easier to describe:

```
pass(1): tcp/connect [: CLIENT PROXY-INT -> PROXY-EXT TARGET
```

This line contains several pieces of information:

”pass”	The request was accepted.
”(1)”	The first SOCKS rule in the server configuration file granted the request access.
”tcp/connect”	The request is a TCP CONNECT request.
”[:”	This signifies that it is the beginning of this TCP session.

The following symbolic names are used in this document as placeholders where values such as hostnames, IP-addresses and port numbers appear in the log messages. For example, in a position where *CLIENT* is listed, an entry such as *127.0.0.1.1044* would appear in an actual log message, as shown in the example above.

”CLIENT”	This entry corresponds to the address and port number of the client making the request to the Dante server.
”CLIENT-UDP”	The address and port number used by a client to send UDP messages to the Dante server.
”PROXY-INT”	This entry contains the internal address of the Dante server, to which clients connect to begin SOCKS protocol negotiation.
PROXY-INTUDP	The address and port number used by the Dante server to receive UDP packets from a client.
”PROXY-EXT”	This entry corresponds to the external address of the Dante server, being the local end of the connections opened to or received from remote hosts.
”NEXTPROXY”	For *server chaining*, the next SOCKS server to which SOCKS requests are forwarded.
”LASTPROXY-EXT”	For *server chaining*, the last endpoint in the chain, from which a connection is made to the TARGET.
”TARGET”	The address of the server or remote endpoint, to which a client requests that a connection should be made to or received from.
”ALLZEROES”	The value *0.0.0.0*. Generally used by the Dante server for values that are unspecified or unknown.
”DESC”	A textual description, usually a platform dependent error message.
”IDLETIME”	A number, indicating the number of seconds a connection has been idle.

3 Log entry field overview

Several of the fields listed above can have different types of values depending on factors such as server configuration and the type of request made by clients.

3.1 Access control

The first part of a traffic log entry shows whether a request is passed or blocked, based on the configuration of the Dante server. The possible combinations are:

”pass(N)”	Traffic or request passed, due to rule N.
”block(N)”	Traffic or request blocked, due to rule N.

3.2 Request types

The SOCKS standard defines several types of requests, translating into several different operations and events that can be controlled by the Dante server:

”tcp/accept”	A TCP connection being received from a SOCKS client (before SOCKS processing begins)
”tcp/connect”	A SOCKS TCP CONNECT request.
”tcp/bind”	A SOCKS TCP BIND request to receive a connection from a remote host.
”tcp/bindreply”	A connection being received from a remote host on a port previously bound with *tcp/bind*.
”udp/udpassociate”	An association created to be able to send and receive UDP packets.
”udp/udpreply”	A reply received to an UDP packet sent to a remote host.

Each request can have several entries in the log file, depending on what the server configuration file specifies. There are typically three types of events that can occur, and each will be marked with a special character combination after the type keyword:

”[:”	First entry for an operation, occurs only once per operation.
”-:”	Traffic being passed, can occur multiple times. Note that this can also include traffic without content, such as when a TCP connection is being closed, or an empty UDP packet is forwarded.
”]:”	Termination of the operation, either due to errors or a connection being closed.

Unless an operation is blocked, the initial ”[:” will always be followed by a terminating ”:]”. If the operation is blocked, no terminating ”:]” will follow.

3.3 Addresses and hosts

Most of the remaining entries contain IP-address and port number pairs for the endpoints that are involved in requests and data transfers. These will typically be in the format of an IP-address, followed by a port number, such as *10.0.0.1.7*, which corresponds to port *7*, IP-address *10.0.0.1*.

In addition to containing only IP-addresses and port numbers, the symbolic values CLIENT, TARGET, etc., can contain alternative or additional data. The SOCKS protocol allows clients to supply symbolic host names in requests, and the Dante server can also include information about user identities in these entries. This gives several possible variations:

”127.0.0.1.1234”	IP-address and port number; here *127.0.0.1* and *1234*.
”rfc931%root@127.0.0.1.1234”	IP-address and user identity, here the user *root* (obtained via *rfc931* authentication), at *127.0.0.1*, port *1234*.
”localhost.1234”	Symbolic hostname and port number, here *localhost*, port *1234*.
”rfc931%root@localhost.1234”	Symbolic host name, port number and user identity; the user *root* (obtained via *rfc931*x authentication), at *localhost*, port *1234*.

Each of the authentication methods supported by Dante has its own symbolic name like *rfc931*, *pam*, etc., and is included for requests that require some form of user authentication.

3.4 Transfer sizes

If logging of I/O transactions is enabled in the Dante server (via the *iooperation* keyword), the size of the number of transmitted bytes will be included at the end of the line. The following line shows an entry for a client sending four bytes to a server:

```
pass(1): tcp/connect -: CLIENT PROXY-INT -> PROXY-EXT TARGET (4)
```

Sending a SIGINFO signal (or SIGUSR1 on platforms without SIGINFO) to the Dante server will also cause a list of ongoing transfers to be listed, along with the total number of bytes transferred in each direction so far:

```
tcp: CLIENT PROXY-INT <-> PROXY-EXT TARGET: idle: 3s,
     bytes transferred: 4 <-> 0
udp: CLIENT-UDP PROXY-INTUDP <-> PROXY-EXT TARGET: idle: IDLETIMEs,
     bytes transferred: 4 <-> 14, packets: 1 <-> 1
```

The example above shows the current state of a connection between CLIENT and TARGET. CLIENT has transmitted a total of four bytes and the connection has been idle for three seconds.

When a connection or UDP association ends, a summary of the number of transmitted bytes will be printed:

```
pass(1): tcp/connect ]: 0 -> CLIENT PROXY-INT -> 4,
                        4 -> PROXY-EXT TARGET -> 0: DESC
pass(1): tcp/accept ]: 0 -> CLIENT PROXY-INT -> 4: DESC
```

In this example, four bytes have been transmitted from CLIENT to TARGET, with zero bytes having gone in the other direction. For *tcp/accept*, only communication between the client and the Dante server is shown. For *tcp/connect*, the data transmitted on both links are shown. Typically, the same number of bytes will be transmitted on both links, but the totals can also be different. For example, if TARGET terminates the connection, the proxy might receive more data from CLIENT than it can forward to TARGET.

The values can also differ if some form of encapsulation is used, such as is the case for UDP, where encapsulation will increase the total number of the bytes exchange between CLIENT-UDP and PROXY-INTUDP beyond the size of the actual payload data.

```
pass(1): udp/udpassociate ]: 14/1 -> CLIENT-UDP PROXY-INTUDP -> 14/1,
                             4/1 -> PROXY-EXT ALLZEROES -> 4/1: DESC
pass(1): tcp/accept ]: 14 -> CLIENT PROXY-INT -> 4: DESC
```

In this example, the SOCKS protocol overhead is 10 bytes. The client sends 14 bytes from CLIENT-UDP to PROXY-INTUDP, but the Dante server forwards only 4 bytes. The remaining 10 bytes come from the SOCKS UDP encapsulation. The value after the slash shows the total number of packets forwarded, in this case 1.

4 Communication transaction types

Each typical networking operation, such as opening or receiving a connection, will generally result in several log entries. This overview should not be considered exhaustive, but it covers the most typical operations that will occur with normal usage. The symbolic names used in the entries below will typically contain IP-addresses followed by port numbers, as specified above, but can also contain symbolic host names and user names, depending on the server configuration.

4.1 Connection blocked before SOCKS request processing

```
block(1): tcp/accept [: CLIENT PROXY-INT
```

A rule blocks connections from CLIENT to the proxy server. No additional entries related to this connection will be logged.

All the examples below show this operation, *tcp/accept*, passing, and the Dante server accepting the connection from the client. A request can still be refused during SOCKS protocol processing, depending on the server configuration.

4.2 Blocked BIND request

```
pass(1): tcp/accept [: CLIENT PROXY-INT
block(1): tcp/bind [: CLIENT PROXY-INT -> PROXY-EXT TARGET
pass(1): tcp/accept ]: CLIENT PROXY-INT
```

A BIND request from CLIENT, expecting a connection from TARGET, is denied.

4.3 Passed BIND request

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(1): tcp/bind [: CLIENT PROXY-INT -> PROXY-EXT TARGET
pass(1): tcp/bindreply [: TARGET PROXY-EXT -> PROXY-INT CLIENT
pass(1): tcp/bindreply -: TARGET PROXY-EXT -> PROXY-INT CLIENT (4)
pass(1): tcp/bindreply -: CLIENT PROXY-INT -> PROXY-EXT TARGET (8)
pass(1): tcp/bindreply -: CLIENT PROXY-INT -> PROXY-EXT TARGET (0)
pass(1): tcp/bindreply ]: 8 -> TARGET PROXY-EXT -> 4,
                          4 -> PROXY-INT CLIENT -> 8: DESC
pass(1): tcp/bind ]: CLIENT PROXY-INT -> PROXY-EXT TARGET
pass(1): tcp/accept ]: 4 -> CLIENT PROXY-INT -> 8: DESC
```

A bind request by CLIENT, followed by a connection made to the bound port PROXY-EXT from TARGET. One TCP connection goes from CLIENT to PROXY-INT and another from TARGET to PROXY-EXT. Four bytes are sent by TARGET and CLIENT replies with an eight byte response before closing the connection. The DESC field provides a textual description of the reason for the connection ending, such as the connection being closed.

4.4 Passed BIND request, blocked incoming connection

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(2): tcp/bind [: CLIENT PROXY-INT -> PROXY-EXT TARGET
block(1): tcp/bindreply [: TARGET PROXY-EXT -> PROXY-INT CLIENT
pass(2): tcp/bind ]: CLIENT PROXY-INT -> PROXY-EXT TARGET
pass(1): tcp/accept ]: CLIENT PROXY-INT
```

CLIENT binds a port on PROXY-EXT, but the server configuration blocks a connection to this port from TARGET.

4.5 Blocked CONNECT request

```
pass(1): tcp/accept [: CLIENT PROXY-INT
block(1): tcp/connect [: CLIENT PROXY-INT -> PROXY-EXT TARGET
pass(1): tcp/accept ]: CLIENT PROXY-INT
```

A request by CLIENT for a TCP connection to TARGET is refused by the Dante server.

4.6 Passed CONNECT request

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(1): tcp/connect [: CLIENT PROXY-INT -> PROXY-EXT TARGET
pass(1): tcp/connect -: CLIENT PROXY-INT -> PROXY-EXT TARGET (4)
pass(1): tcp/connect -: TARGET PROXY-EXT -> PROXY-INT CLIENT (4)
pass(1): tcp/connect -: CLIENT PROXY-INT -> PROXY-EXT TARGET (0)
pass(1): tcp/connect ]: 4 -> CLIENT PROXY-INT -> 4,
                        4 -> PROXY-EXT TARGET -> 4: DESC
pass(1): tcp/accept ]: 4 -> CLIENT PROXY-INT -> 4: DESC
```

A connection request is made by CLIENT to PROXY-INT. The Dante server opens a connection from PROXY-EXT to TARGET. The client sends a four byte message and receives a four byte response. The client then closes the connection.

4.7 CONNECT request passed via *server chaining*

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(1): tcp/connect [: CLIENT PROXY-INT ->
                        PROXY-EXT NEXTPROXY LASTPROXY-EXT TARGET
pass(1): tcp/connect -: CLIENT PROXY-INT ->
                        PROXY-EXT NEXTPROXY LASTPROXY-EXT TARGET (4)
pass(1): tcp/connect -: TARGET LASTPROXY-EXT NEXTPROXY PROXY-EXT ->
                        PROXY-INT CLIENT (4)
pass(1): tcp/connect -: CLIENT PROXY-INT ->
                        PROXY-EXT NEXTPROXY LASTPROXY-EXT TARGET (0)
pass(1): tcp/connect ]: 4 -> CLIENT PROXY-INT -> 4,
                        4 -> PROXY-EXT NEXTPROXY LASTPROXY-EXT TARGET -> 4
pass(1): tcp/accept ]: 4 -> CLIENT PROXY-INT -> 4: DESC
```

Having a SOCKS server forward a request to another SOCKS server is called server chaining. The next-hop SOCKS server can again use SOCKS to forward the same request to a third SOCKS server, and so on. A SOCKS server using server chaining cannot know each hop the connection makes, it can only know the identity of the next SOCKS server and the external IP-address and port number of the last hop before the destination (TARGET). Note that the Dante server also supports chaining through non-SOCKS proxies (e.g., HTTP proxies), which might not make the LASTPROXY-EXT value available to the Dante server.

To be able to identify a client from a single log file on the TARGET end, it is necessary to include additional information during server chaining. When server chaining is used by the Dante server, the log entries also specify the IP-address and port number of the next SOCKS server in the chain (NEXTPROXY), and the external IP-address and port number of the last SOCKS server (LASTPROXY-EXT). Instead of two addresses for each connection endpoint, information on the endpoints of three TCP connections are included.

The transaction above shows CLIENT making a CONNECT request for TARGET. The Dante server forwards the request via NEXTPROXY, which is another SOCKS server. The TCP connection for the last hop goes from LASTPROXY-EXT to TARGET. The client sends a four byte request, receives a four byte response and then closes the connection.

4.8 CONNECT request timeout

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(1): tcp/connect ]: CLIENT PROXY-INT -> PROXY-EXT TARGET: Timeout
pass(1): tcp/accept ]: CLIENT PROXY-INT
```

The above log output shows a TCP CONNECTION request being made by CLIENT for TARGET, which never completes due to a timeout. The timeout message is platform dependent. The SOCKS server closes the connection to the client when the timeout occurs. Other communication failures will result in similar log entries with different error descriptions.

4.9 Blocked UDP ASSOCIATE request

```
pass(1): tcp/accept [: CLIENT PROXY-INT
block(0): udp/udpassociate [: CLIENT-UDP ALLZEROES -> PROXY-EXT ALLZEROES
pass(1): tcp/accept ]: CLIENT PROXY-INT
```

The client requests a UDP association, but this is refused by the Dante server.

4.10 Passed UDP ASSOCIATE request

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(1): udp/udpassociate [: CLIENT-UDP PROXY-INTUDP -> PROXY-EXT ALLZEROES
pass(1): udp/udpassociate -: CLIENT-UDP PROXY-INTUDP -> PROXY-EXT TARGET (4)
pass(1): udp/udpreply -: TARGET PROXY-EXT -> PROXY-INTUDP CLIENT-UDP (4)
pass(1): udp/udpassociate ]: 14/1 -> CLIENT-UDP PROXY-INTUDP -> 14/1,
                             4/1 -> PROXY-EXT ALLZEROES -> 4/1: DESC
pass(1): tcp/accept ]: 14 -> CLIENT PROXY-INT -> 4: DESC
```

The UDP exchange above shows CLIENT establishing a UDP association, sending one UDP packet to TARGET, and then receiving UDP packet back. The payload is four bytes in both exchanges; the higher values in the final total include the overhead of sending UDP packets via SOCKS, the lower values show only the total payload sizes (four bytes) and packet count (one packet).

4.11 Passed UDP ASSOCIATE request, blocked *udpreply*

```
pass(1): tcp/accept [: CLIENT PROXY-INT
pass(2): udp/udpassociate [: CLIENT-UDP PROXY-INTUDP ->
                             PROXY-EXT ALLZEROES
pass(2): udp/udpassociate -: CLIENT-UDP PROXY-INTUDP ->
                             PROXY-EXT TARGET (4)
block(1): udp/udpreply -: TARGET PROXY-EXT ->
                          PROXY-INTUDP CLIENT-UDP (4)
pass(2): udp/udpassociate -: CLIENT-UDP PROXY-INTUDP ->
                             PROXY-EXT TARGET (4)
block(1): udp/udpreply -: TARGET PROXY-EXT ->
                          PROXY-INTUDP CLIENT-UDP (4)
pass(2): udp/udpassociate -: CLIENT-UDP PROXY-INTUDP ->
                             PROXY-EXT TARGET (4)
block(1): udp/udpreply -: TARGET PROXY-EXT ->
                          PROXY-INTUDP CLIENT-UDP (4)
pass(2): udp/udpassociate ]: 0/0 -> CLIENT-UDP PROXY-INTUDP -> 42/3,
                             12/3 -> PROXY-EXT ALLZEROES -> 12/3: DESC
pass(1): tcp/accept ]: 0 -> CLIENT PROXY-INT -> 12: DESC
```

The Dante server allows CLIENT to establish an UDP association from CLIENT-UDP. The Dante server uses PROXY-INTUDP to receive UDP packets from the client. A packet is sent to TARGET by the client, but all replies are blocked by the SOCKS server. The client retries three times before giving up and ending the association.