Dante – a BSD licensed SOCKS implementation

Inferno Nettverk A/S

Bergen Linux User Group

2005-11-24

Background (1)

Inferno Nettverk A/S started as security consulting company

- goal: sell OpenBSD based firewalls/security solutions.
- packet filter existed in OpenBSD, wanted general proxy in addition.
- SOCKS5 implementation commercial, original V4 with free license.
- plan: add support for V5 of socks standard to free implementation.
- work started 1997/09/26 12:35:33.
- first release on the 16th of Nov 1998 (version 0.90.0).

Background - first announcement

```
Subject: Dante - a free socks client and server implementation for UNIX
To: socks@socks.nec.com
Date: Mon, 16 Nov 1998 15:30:04 +0100
X-Mailer: Mutt 0.89.1

Inferno Nettverk A/S, Oslo, Norway
Monday, November 16, 1998

Inferno Nettverk is pleased to announce the first public alpha release
of Dante - a free socks client and server implementation for UNIX.

It currently supports v4 and v5 (sans GSSAPI) of the socks protocol
and is available under a BSD/CMU-type license with complete source code.

Dante can be downloaded from ftp.inet.no:/pub/socks/danta-alpha.tar.gz.

One should note that Dante is currently somewhat, to put it politely,
lacking in documentation, but we are working on that too.

For more information, please see http://www.inet.no/dante.

(This implementation includes a small experimental extension to the
v4/v5 protocol that provides a more generic bind functionality, such
as that which might be expected by server applications like e.g ftpd.
It can be enabled by providing the magic line "extension: bind" at the
correct place in the server and client config file.)
```

Today

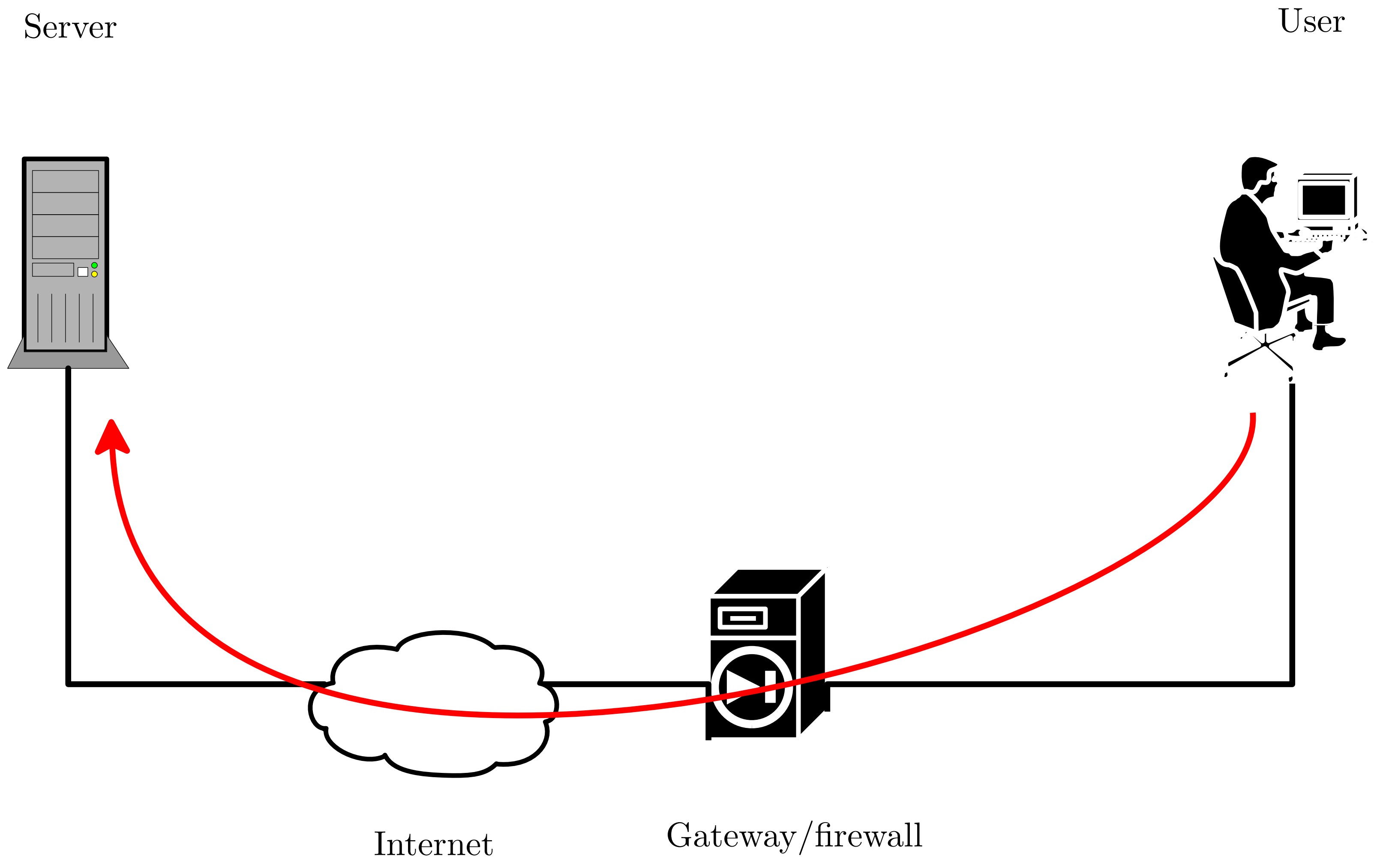
Inferno Nettverk A/S no longer limits itself to security consulting, but also offers general UNIX-based development, from web-programming to low-level network programming.

Dante

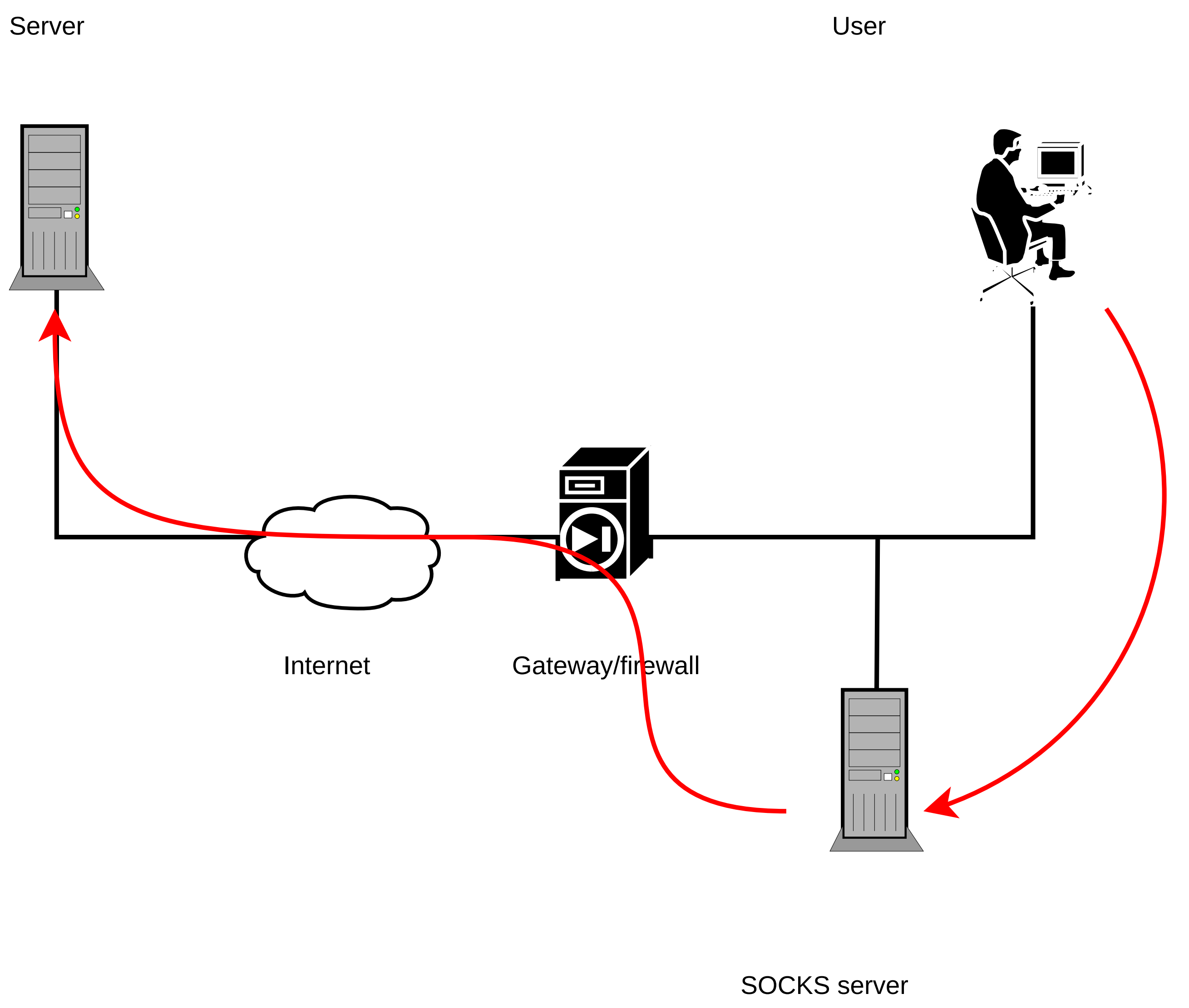
- ended up being a total rewrite.
- over 20 releases so far (excluding prereleases).
- has been described as scalable and stable by users.
- many large international companies among users.

SOCKS

Connection establishment - normal network



Connection establishment - SOCKS network



SOCKS protocol - overview (1)

- general proxy protocol.
- version 5 defined in rfc1928.
- client hosts do not make connections directly, but via SOCKS server.
- hides internal network (ala NAT).
- authentication possible (PAM, username/password).

SOCKS protocol - overview (2)

Commands: CONNECT, BIND, UDP ASSOCIATE

- outgoing TCP connections (CONNECT).
  — e.g. HTTP
- outgoing UDP packets, and reply (UDP ASSOCIATE).
  — e.g. DNS
- incoming TCP connections, originaly from known IP/port number (BIND), but today interpreted more lose.
  — typically *active* FTP

Other

- bind extension.
- GSSAPI (unsupported).

SOCKS protocol - overview (3)

Why use SOCKS?

- user authentication (allows per user resource accounting).
- avoids having traffic from bad TCP/IP implementations on the Internet.
- firewalls good for regulating incoming traffic, SOCKS for outgoing.

Dante configuration - server (1)

Dante server rules operate on two levels.

client-rules Rules prefixed with "client". Control access to the Dante server at the TCP level. Authentication possible here too (e.g. PAM or ident).

socks-rules Control access via the Dante server, operating at the socks protocol level.

Simple configuration; pass in/out all traffic.

```
internal: fxp0 port = 1080
external: fxp1
method: none

client pass { from: 0.0.0.0/0 to: 0.0.0.0/0 }
pass { from: 0.0.0.0/0 to: 0.0.0.0/0 }
```

Dante configuration - server (2)

Add user authentication.

As above, but change "method: none" to "method: username".

Dante configuration - server (3)

Limit access based on user/IP-address. As above, but create individual rules for each user/IP-address group instead of one global:

```
block { from: 0.0.0.0/0 to: .naughty.com port = http }

pass { from: 0.0.0.0/0 to: 0.0.0.0/0 port = http
       user: big-boss
       log: connect disconnect }

pass { from: 0.0.0.0/0 to: 0.0.0.0/0
       user: awk }

pass { from: 0.0.0.0/0 to: 0.0.0.0/0 port = http
       maxsessions: 10
       bandwidth: 102400 }

pass { from: 0.0.0.0/0 to: 0.0.0.0/0 }
```

Dante configuration - client (1)

Clients wishes to establish a connection to a given address/port. This cannot be done directly, request must be done via SOCKS server.

- scenario 1: Client support for SOCKS standard
  - configure client to use server (e.g. browser)
- scenario 2: *socksify*
  - allows (dynamically linked) programs to use socksify without changes.
  - LD_PRELOAD based approach, works on most UNIX platforms.
  - client part of SOCKS protocol in Dante *libdsocks* library.
  - system calls intercepted, connections made via SOCKS server.
  - client configuration in global *socks.conf*.

Dante configuration - client (2)

- scenario 3: windows
  - — program which *socksifies* the entire system exists. Not from Inferno Nettverk.
- scenario 4: no preload support
  - — applications must be recompiled if possible.

Dante configuration - client (3)

Client-configuration simpler, only a route statement required.

```
route { from: 0.0.0.0/0 to: 0.0.0.0/0 via: 10.1.1.1 port = 1080
  proxyprotocol: socks_v5 socks_v4
  proxyprotocol: http_v1.0
  proxyprotocol: msproxy_v2
  method: none username
}
```

Multiple routes might exist, client will try first working one.

Dante modules

Dante partially financed by support contracts. The *module* concept is an experiment in selling extra functionality.

At the moment three modules exist.

- redirect - redirection of connections and control over address/port-usage.
- bandwidth - bandwidth limitation.
- session - limit sessions (not official yet).

Experiences

Dante originally developed on OpenBSD, then ported to various UNIX platforms.

- before: nice and elegant code, after: spaghetti of #ifdef’s.
- have tried to limit this, by adding missing functionality in library.
- have found many OS bugs, often in Linux and Solaris (2.5 especially).
- bugs in OpenBSD easy to fix, must work around in many other cases.
- preloading especially difficult, header files contain much strangeness (esp. Linux and OSF).

Development

Much work this year on testing architecture.

- runs various applications with and without SOCKS, compares result.
- automated testing, different configuration files for each test.
- several bugs found, mainly in rarely used parts of the code.
- example: client support for http proxy servers; errors were not handled correctly.
- main benefit is catching bugs introduced during development.
- currently used internally only, generalisation is long term goal.
- partial support for server chaining new feature in next version.

Possible extensions

Routers and firewalls see packets, while all traffic passes through SOCKS server as a sequential stream. Makes some creative usages of the SOCKS server potentially possible.

- inspection of data stream; identification of downloaded data types.
- protocol decoding, and identification of application.
- possible usage: blocking traffic from P2P networks.

Future

Initially many new features introduced. System considered stable for the last couple of years, no big changes made.

Current focus is on bug fixing and maintainance.