

**NAME**

**knc** — kerberized netcat

**SYNOPSIS**

```
knc -l [-n] [-d] [-a bind_address] [-f] [-c num] port prog [args]
knc -il [-n] [-d] prog [args]
knc -ls path [-n] [-d] [-a bind_address] [-f] [-c num] port
knc -ils path [-n] [-d]

knc [-d] [-n] service@host port
knc [-d] [-n] -N fd service@host
```

**DESCRIPTION**

**knc** provides an 8-bit clean, mutually authenticated tunnel between two endpoints. The same executable provides both client and server functionality.

The server can operate in either "inetd" or standalone mode. In server mode, **knc** either launches *prog* with arguments *args* or connects to a UNIX domain socket (depending on the presence of the **-S** flag).

The options are as follows:

- l** listener (server) mode.
- i** set "inetd" mode.
- n** do not use the resolver for any name look ups (no DNS mode).
- d** increment debug level (specify multiple times for increased debugging).
- a** *bind\_address* bind to address *bind\_address* when in server mode (default is INADDR\_ANY).
- S** *path* connect to the named Unix domain socket upon accepting a connection rather than launching a program.
- f** don't fork when in server mode (useful for debugging).
- c** in server mode, limit the maximum number of child processes to *num*.
- w** in server mode, start as an inetd wait service. That is, expect stdin to be a listening socket and process requests on it.
- M** *max* in server mode, the maximum number of connexions to process before exiting.
- N** *fd* in client mode, do not attempt to connect to a remote host, but instead use the supplied, pre-connected file descriptor *fd*. The usual knc handshake will be performed over this file descriptor.
- P** *sprinc* in client mode, specify the Kerberos principal that we will use for the server.
- S** *sun\_path* in server mode, connect to the UNIX domain socket specified by *sun\_path* rather than run a program.
- T** *max\_time* in server mode, the maximum time to process requests.

When **knc** launches a program, it inserts the principal of the counter-party into the environment variable KNC\_CREDS as well as populating other environment variables. (See **ENVIRONMENT AND UNIX DOMAIN SOCKET PROTOCOL**)

The server connects its network side to the stdin and stdout file descriptors of the launched program. Any reads or writes by the launched program are translated into reads and writes to the network side. Likewise,

reads and writes on the network side are translated to the local side. End of file conditions (EOF) are similarly translated.

Similarly, the client connects its stdin and stdout file descriptors to its network side, translating reads and writes as above.

## ENVIRONMENT AND UNIX DOMAIN SOCKET PROTOCOL

**knc** has two distinct ways of communicating information to the server-side process. If **knc** is launching an executable, it communicates by populating the environment of the launched program. However, if **knc** is instead connecting to a Unix domain socket, it must transmit the same information over the socket to the server process.

For launched executables, the current environment variables are defined:

KNC_CREDS	The full principal of the remote counterparty.
KNC_REMOTE_IP	The IP address of the <b>knc</b> client program. N.B. <i>NO ENTITLEMENT DECISIONS</i> should be based on the contents of this variable. Further, it is only the "nearest" client to the server. Remember that various other tunnels (including <b>knc</b> ) may be between you and the actual user.
KNC_REMOTE_PORT	The source port of the client.
KNC_VERSION	The version of the server. This is not the version of the client as the server does not know this.

When **knc** instead connects to a UNIX domain socket, it uses the following protocol to transmit the information contained in the environment variables:

```
Key_1:Value_1\n
Key_2:Value_2\n
...
END\n
```

These *KEY:VALUE* pairs will be the very first data transmitted across the newly accepted Unix domain socket. Currently defined *KEYs* are precisely the same as the environment variables detailed above, without the KNC\_ prefix. (e.g. *CREDS*, *REMOTE\_IP*, etc.)

The server application must parse this protocol until the *END\n* indicator is seen. The application is free to ignore any of the *KEY:VALUE* pairs it sees.

Once these have been transmitted, **knc** begins relaying data as normal. No acknowledgement on the part of the server application is required, and further, it is prohibited, as this will be counted as part of the normal data stream.

## SECURITY CONSIDERATIONS

Use of **knc** must be carefully considered in order to bring security benefits to your application. In particular, applications launched by **knc** which wish to trust the contents of KNC\_CREDS must not allow themselves to be executed by any means other than **knc**. One method of ensuring this is to cause the launched program to be owned and executable only by a special-purpose uid which issues the **knc** command.

## DETAILS AND APPLICATION CONSIDERATIONS

A typical **knc** deployment looks like the diagram below:



