NonStop SSL Server (NSSL)

Crystal Point

Version 3.3

Document History:
Document version 3.3 issued April 21, 2009

This manual was produced by:

Crystal Point
19515 North Creek Parkway #204
Bothell, WA 98011
USA

Contents

Preface

Who Should Read this Guide

This document is for system administrators or web masters who are responsible for installing and configuring the NonStop SSL Proxy (NSSL) to secure Telnet sessions in conjunction with OutsideView or other telnet client. Instructions are also provided for configuring NSSL as a secure proxy for FTP.

As NSSL can also be used as a standalone NonStop web server, this guide also instructs web masters how any HTTP content can be served by the NSSL HTTP server.

Those interested in details of how NSSL handles certificates and how certificates can be generated using the Tools supplied by Crystal Point within NSSL, can also find that information in this document, as well.

Document History

Version 3.3

- Contains information corresponding to NSSL Version 1058 or later
- The new parameters AUDITASCIIONLY, AUDITASCIIDUMPLENIN, AUDITASCIIDUMPLENOUT, FTPLOCALDATAPORT, INTERFACE, TARGETINTERFACE have been documented
- The new cipher suites 0.1 and 0.2 have been documented
- The parameters which can be changed with the SET command of NSSLCOM have been updated

Version 3.2

- Contains information corresponding to NSSL Version 1054 or later
- The new parameters DONOTWARNONERROR, CONTENTFILTER, MAXSESSIONS, SOCKSHOST, SOCKSPORT and SOCKSUSER have been documented
- The NSSLCOM STATUS command now displays the number of sockets as well as the total CPU usage of NSSL
- The OpenSSL library was updated from OpenSSL 0.9.7d to 0.9.8b
- The new NSSLCOM command CONNECTIONS, STATS is documented in chapter "Monitoring NSSL"
- The extended SHOWLOG syntax allowing "filtering by timestamp" is documented in the chapter "Monitoring NSSL"
- The parameters ALLOWIP and DENYIP can now be configured to only apply to one direction, this has been documented

Version 3.1

- Contains information corresponding to NSSL Version 1051 or later
- The section "Monitoring NSSL" has been rewritten.
- The table of TLS alerts has been moved to the "SSL Reference" section.
- The documentation for the parameters PEERCERTCOMMONNAME and PEERCERTFINGERPRINT has been corrected.
- An error in the documentation of the PASSIVE parameter has been fixed.

Version 3.0

- Contains information corresponding to NSSL Version 1047 or later
- Execution of NSSLCOM commands can now be restricted, using the new parameter NSSLCOMSECURITY.
- The number of licensed CPU's for each Itanium system may now be included in the license file.
- The new NSSLCOM command SSLINFO will display the local certificate chain.

- The new NSSLCOM command RELOAD CERTIFICATES allows for the changing of the server certificate chain without having to restart NSSL.
- The new parameters LOGEMS, LOGFORMATEMS, LOGLEVELEMS enable logging to EMS.
- The new parameters PEERCERTFINGERPRINT and PEERCERTCOMMONNAME support verification of remote certificates.
- The documentation for the CLIENTAUTH parameter was misleading and has been rewritten.

Version 2.9

- Contains information corresponding to NSSL Version 1046 or later
- Considerations for installing on Itanium systems have been added in section "Considerations for installing on different NonStop server versions" in chapter "Installation".
- The "rollover" of both LOG and AUDIT files has been improved. This is reflected in numerous locations within the manual, notably:
    - see new parameters LOGFILERETENTION and AUDITFILERETENTION
    - see new NSSLCOM command ROLLOVER LOGFILE
    - see new section "Logfile/Auditfile rollover using round robin" in chapter "Configuring And Running NSSL".
- The formatting of the two different "targets" for log messages (LOGFILE and LOGCONSOLE) is more flexible now. Please see the new parameters LOGFORMATFILE, LOGFORMATCONSOLE, LOGLEVELFILE, LOGLEVELCONSOLE as well as the new NSSLCOM command LOGMESSAGE
- The new parameters TCPIPHOSTFILE, TCPIPNODEFILE and TCPIPRESOLVERNAME allow the setting of DEFINEs from the runtime parameter string or configuration file.
- A new parameter TCPNODELAY allows to control activating RFC1323 on the sockets
- References to external documents on SSL have been updated

Version 2.8

- Contains information corresponding to NSSL version 1045 or later.
- discusses new feature of auditing plain or secure FTP traffic in FTPS mode:
    - see sections "NSSL as a plain FTP Server Proxy " in chapter "Introduction" and "To start the NSSL FTP Server Proxy with an audit log" in chapter "Installation" for an overview.
    - see new parameters AUDITLOG, AUDITCONSOLE, AUDITFORMAT, AUDITLEVEL, AUDITMAXFILELENGTH, FTPALLOWPLAIN for details on the new parameters.
- discusses new parameter FTPCALLOW200REPLY which switches on compatibility to some older FTP/TLS servers.
- discusses new parameter LOGLEVELFILE which allows to set LOGLEVEL to different values for output to LOGCONSOLE and LOGFILE.

- discusses new parameter CONFIG2 which allows to configure a second configuration file.

Version 2.7

- Contains information corresponding to NSSL version 1044 or later.
- The documentation of the parameter ALLOWCERTERRORS has been extended.
- The new run mode FTPCPLAIN has been added.
- The new run mode EXPANDS for encryption of Expand over IP traffic has been added.
- The new run mode ODBCMXS for encryption of ODBC/MX traffic has been added.

  Functional enhancements of NSSLCOM have been documented:

  - The STATUS command now displays more information
  - The parameters ALLOWIP,DENYIP,TRUST and ALLOWCERTERRORS can now be changed through NSSLCOM
- The new parameter PTCPIPFILTERKEY is documented.
- Support of the new MIME type "jnlp" is documented.
- The new parameter CLIENTAUTH and the changed functionality of the parameter TRUST are documented.

Version 2.6

- Contains information corresponding to NSSL version 1040 or later
- New commands CONNECTIONS, INFO CONNECTION [,DETAIL], RENEGOTIATE LPORT and SET LOGMEMORY have been added to NSSLCOM

Version 2.5

- Contains information corresponding to NSSL version 1036 or later
- The OpenSSL library was updated from openssl 0.9.7a to 0.9.7d
- Performance optimization:
  - about 30 % bis 55 % less CPU usage for RSA handshake
  - about 35 % less CPU usage for 3DES bulk encryption
  - Chapter "Performance Considerations" was updated with new measurement data.
- A new chapter about "SSL Client Authentication checking" when running in server mode was added.
- New parameter ALLOWCERTERRORS

Version 2.4

- A new chapter about "SSL Client Authentication" has been added.
- Documentation for the new run modes MQS and MQC to support Websphere MQ.
- The parameters KEEPALIVE, CLIENTCERT, CLIENTKEY, CLIENTKEYPASS have been added..

Introduction

What Is the NSSL Server?

The NonStop SSL (NSSL) Server is a server program for the HP NonStop Guardian platform designed to cover a lot of different usage scenarios. NSSL can meet the following requirements:

- [HTTP] Acting as an easy-to-configure web server for the NonStop Guardian platform which can be used to serve OutsideViewWEB via the HTTP protocol to a standard network browser.
- [HTTPS] Acting as a *secure* web server supporting the HTTPS protocol to be able to deploy OutsideViewWEB in a secure fashion.
- [TELNETS] Acting as a secure proxy server for the NonStop TELSERV, to secure the communication between the NonStop system and a secure telnet client, such as OutsideView.
- [PROXYS] Acting as a secure proxy server for plain TCP/IP servers acting as Server Gateways for Client/Server-Middleware, such as the NonStop RSC product.
- [PROXYC] Acting as a client proxy for plain TCP/IP client programs, to secure the communication between the NonStop Server and remote SSL-enabled server programs.
- [FTPS] Acting as a secure proxy server for plain FTP servers, such as the NonStop FTPSERV, to secure the communication between the NonStop system and a secure FTP client, such as OutsideView version 7.3 or above, or with any FTP client which is SSL-enabled by the Remote Proxy available from Crystal Point.
- [FTPC] Acting as a client proxy for the NonStop FTP client program, to secure the communication between the NonStop system and a secure FTP server, such as the WS_FTP Server, or with any FTP server which is SSL-enabled by the Remote Proxy available from Crystal Point.
- [FTPCPLAIN] Acting as a "plain-to-plain client proxy" for the NonStop FTP client program, to enable unencrypted FTP file transfers with PASSIVE mode.
- [ATTUNITYS] Acting as a server proxy for the NonStop Attunity program, to secure the communication between the NonStop system and a remote Attunity Client. The remote client will be secured using the Remote Proxy component.

- [MQS] Acting as a server proxy for the WebSphere MQ program, to secure the communication between the non secure 5.1 release on the NonStop system and a secure remote WebSphere MQ Client (5.3).
- [MQC] Acting as a client proxy for the WebSphere MQ program, to secure the communication between the non secure 5.1 release on the NonStop system and a secure remote WebSphere MQ Server (5.3).
- [EXPANDS] Creating an SSL tunnel to secure EXPAND over IP lines.
- [ODBCMXS] Acting as a server proxy for the ODBC/MX protocol.

To support the above functions, NSSL can be started in different modes. These so-called "run modes" of NSSL are listed in square brackets in the list above. Multiple NSSL processes can co-exist on a single NonStop system to support concurrent (secure and non-secure) web and proxy services, as well as multiple TCP/IP processes.

NSSL is delivered with a license file which will determine the allowed run modes of NSSL. Without a license file, NSSL will only run as a Web server NOT capable of SSL ("HTTP" mode). Note that this run mode ("plain web server") is the only run mode where NSSL will not use SSL.

The following table lists all run modes of NSSL:

Run Mode	**Usage**
FTPC	FTP client proxy
FTPS	FTP server proxy
FTPCPLAIN	FTP client proxy without encryption
HTTP	HTTP server
HTTPS	Secure HTTP server
PROXYS	Generic SSL server proxy
PROXYC	Generic SSL client proxy
TELNETS	Secure Telnet proxy
ATTUNITYS	Secure Attunity proxy
MQS	Secure WebSphere MQ server proxy
MQC	Secure WebSphere MQ client proxy
EXPANDS	Secure EXPAND proxy
ODBCMXS	Secure ODBC/MX proxy

NSSL Features

SSL

NSSL uses SSL (Secure Socket Layer) in the TLS (Transport Layer Security) variant as standardized by the IETF in RFC 2246, to secure an application on the transport layer. SSL 2.0, SSL 3.0 and TLS 1.0 (SSL 3.1) are supported. NSSL offers multiple configurable cipher suites with RSA key exchange with public key lengths up to 2048 bits and up to 168 bit Triple DES for bulk encryption.

Non-Stop Availability

Using NSSL ensures non-stop availability of NonStop based applications across the network. Running on the Guardian platform, NSSL takes advantage of the NonStop fundamentals.

On G series systems, NSSL services can be configured as generic processes, enabling automatic recovery from failures, such as CPU outages. For D series systems, non-stop availability can be achieved by implementing NSSL services as static PATHWAY servers monitored by a non-stop Pathway Monitor.

Parallel Library Support

NSSL fully supports the NonStop Parallel Library TCP/IP. Please see the parameter SUBNET in the parameter reference part for configuration details.

Performance

NSSL is optimized for performance on the NonStop Guardian platform. The number of concurrent sessions (for both HTTP or proxy sessions) is virtually only limited by the system's resources. Running as a Telserv proxy, NSSL can easily handle the maximum number of 255 sessions supported by a single Telserv process.

The performance impact of a single unencrypted terminal session is minimal. Tests have shown that the additional load generated by an NSSL proxy for tunneling the data (without encrypting) is negligible (a "FILEINFO *.*" showed an increase in overall CPU busy time of all relevant processes smaller than 3%).

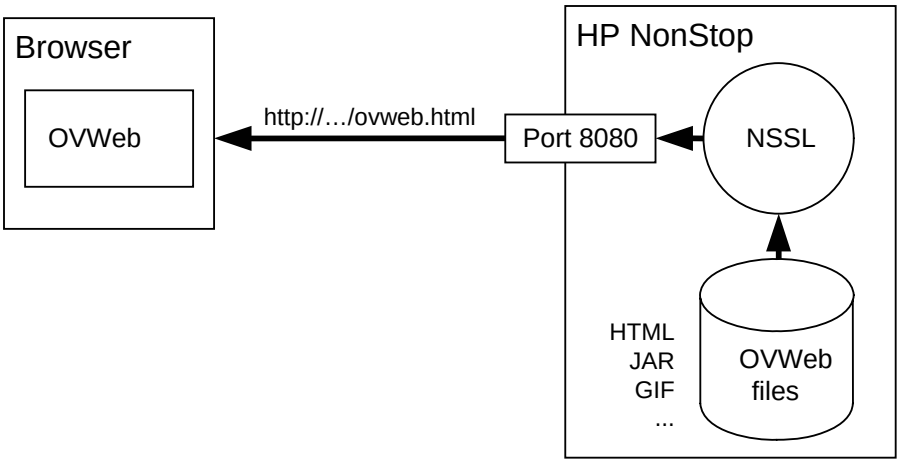
The performance impact is larger for the FTPS mode, as much more data per second is sent over the network.

Please see the "Performance Considerations" chapter for a detailed performance analysis of SSL sessions.

NSSL as a Web Server

Running as a HTTP server, NSSL provides the functionality required to download OutsideViewWEB or AppView applet files with standard browsers

like Microsoft Internet Explorer or Netscape Navigator. The certificate tools may also be served by NSSL executing as a HTTP server.



NSSL as a HTTP server downloading OutsideViewWEB to a browser.

Unlike a full blown web server, NSSL is very easy to configure. This avoids the configuration and management overhead of a standard web server if one does not already exist. Last but not least NSSL supports the native Guardian platform, enabling deployment of OutsideViewWEB or AppView without having to install OSS on the NonStop server or introduce a separate web server machine representing a potential point of failure.

Overcoming Guardian File System Restrictions

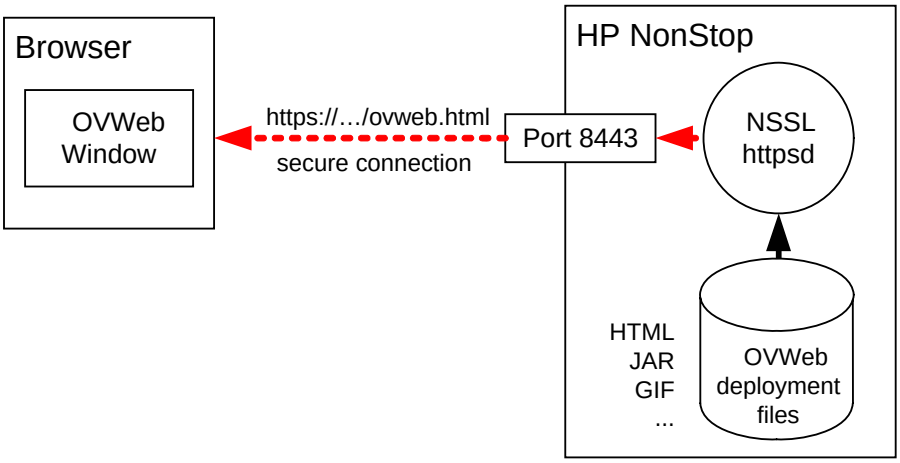
The Guardian file system only supports file names with 8 characters without filename extensions. Furthermore, it is a "flat" file system. Although allowing to group files in "subvolumes", it does not provide a multi-level directory hierarchy like Windows or Unix. These restrictions can be an obstacle when deploying HTTP contents with a Guardian-based web server. HTTP contents is usually grouped in hierarchical structures and makes heavy use of long file names and extensions (e.g. .html, .gif, .jpg, .jar).

To overcome the restrictions of the Guardian file system, NSSL can serve HTTP contents from a standard ZIP file containing files with their full (long) path names. Thus, the HTTP contents can be easily developed and organized on a standard workstation. For deployment with NSSL the required files simply need to be packed with a standard ZIP tool and transferred to the NonStop server.

NSSL as a Secure Web Server

NSSL also supports the HTTPS protocol (when a license that enables that feature is installed). HTTPS means that the HTTP session is encrypted using the SSL (Secure Sockets Layer) protocol.

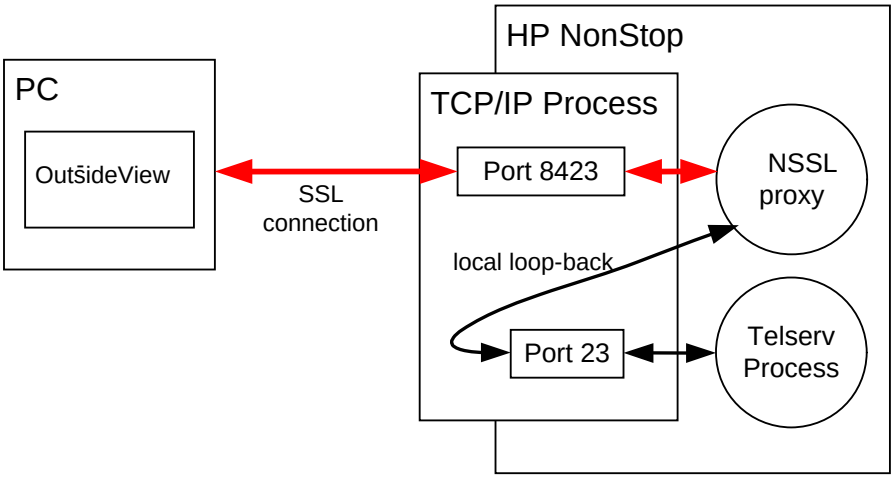
The figure below illustrates how documents may be securely downloaded to an SSL capable browser. The browser connects to an https URL, indicating that the requested resource should be downloaded securely. On the NonStop server side, NSSL will accept the connection, perform the SSL handshake to setup a secure session and encrypt the requested resource when sending it to the browser.



NSSL acting as a HTTPS server for securely downloading OutsideViewWEB to a browser

NSSL as a Secure Proxy for Telnet Access

NSSL can be run as a proxy process to front-end TCP/IP servers accepting plain TCP connections, such as the NonStop Telserv process. With its SSL support, NSSL will enable secure communication to clients, which also support the SSL protocol, such as OutsideView.



NSSL as an SSL proxy front-ending the standard NonStop Telserv process

Acting as a proxy server, NSSL will accept SSL connections from the network and "tunnel" them to a plain TCP server. Encrypted data received from the SSL client will be decrypted and forwarded to the server. Plain data received from plain TCP server will be encrypted and sent to the SSL client. For example, from the Telnet server's point of view the proxy acts as a normal Telnet client, while from an SSL telnet client the NSSL proxy authenticates the Telnet server and encrypts/decrypts the session's payload.

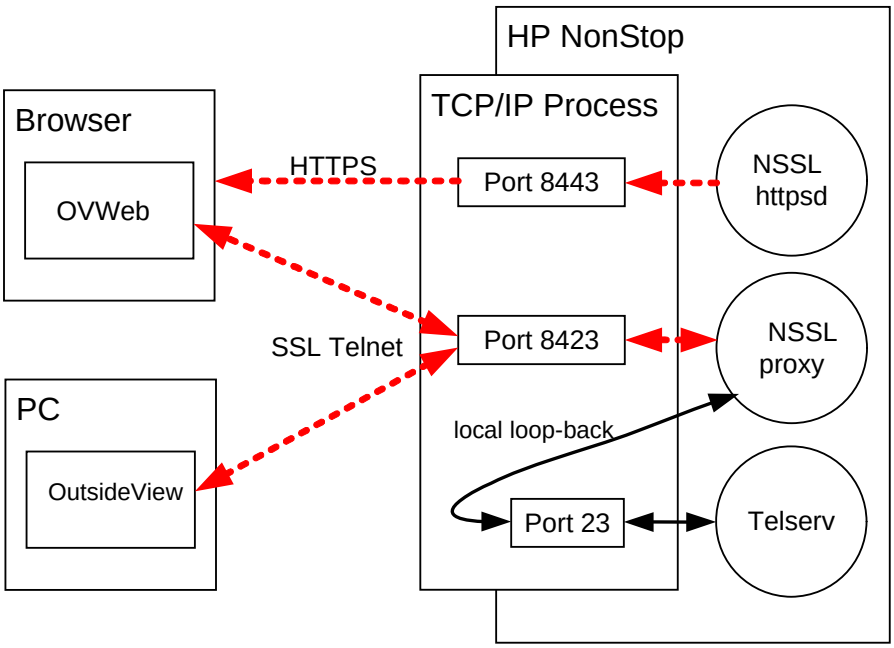
Typically, an NSSL proxy will reside on the same IP process on the same system as the TCP server it tunnels the session to, which allows creation of a "local loopback" session (a connection to "127.0.0.1") for the unencrypted

data. This avoids having any unencrypted data traverse the network. For a local loopback, the data is only being passed within the local TCP/IP stack.

One instance of an NSSL proxy handles SSL connections received on a single IP process and port number and tunnels them to a single target port. Hence if multiple plain ports need to be secured, such as multiple Telnet Servers, an NSSL process needs to be started for each plain TCP port.

Secure Telnet Access Overview

Combining NSSL's functionality as a HTTPS web server and SSL proxy will allow you to completely secure your NonStop Telnet application access. Client workstations running SSL-enabled terminal emulation software such as OutsideView will access Telserv through the NSSL Secure Proxy. Browser-based solutions with SSL telnet support (e.g. secure OutsideViewWEB or AppView) accessing NonStop based applications, can be deployed directly on the host the application is running on, without installing OSS on your Guardian machine or installing a separate web server.



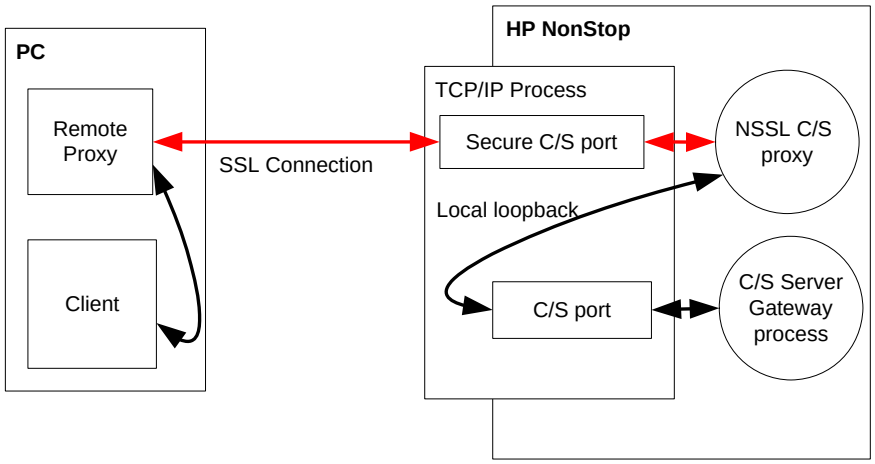
Secure application access with NSSL HTTPS server and SSL proxy

For a secure browser-based solution, you will need at least two NSSL processes, one running as HTTPS server and another running as secure Telserv proxy, as shown in the figure above. The OutsideViewWEB or AppView applet will be securely downloaded using the NSSL HTTPS server. Once initialized, the applet will initiate a secure telnet session via the NSSL proxy.

NSSL as a Secure Proxy for Generic TCP/IP Client/Server Applications Access

An NSSL proxy allows SSL encryption not only for Telnet but for any Client/Server protocol facilitating TCP sockets. While NSSL acts as a secure

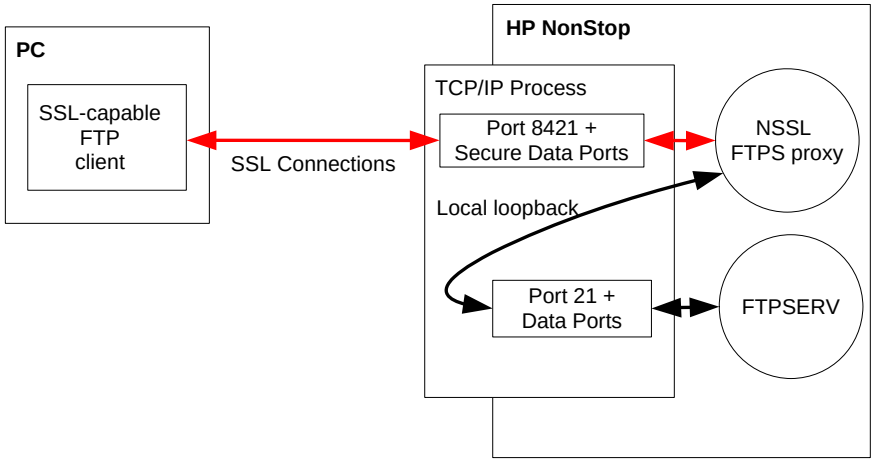
proxy for the server-side of the client/server communication, the client-side may be enabled for SSL using the Remote Proxy available from Crystal Point.



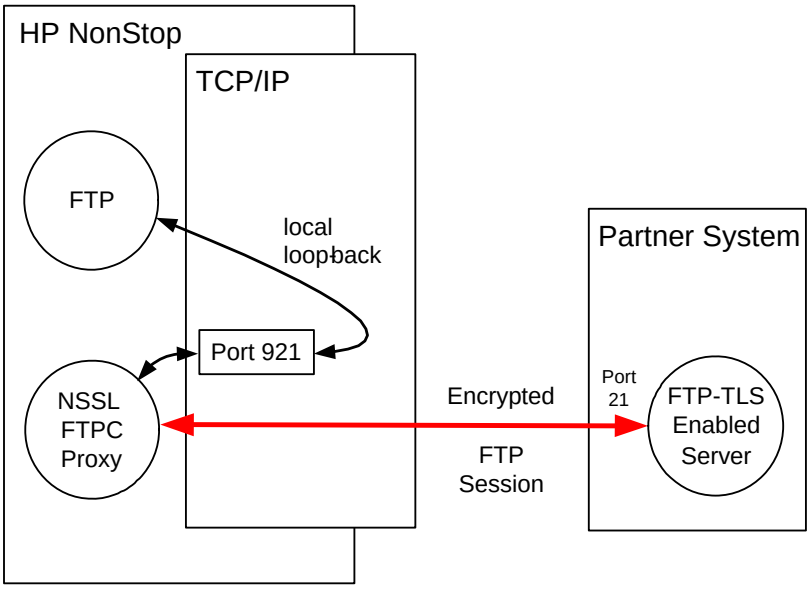
NSSL Client/Server proxy front-ending a server gateway process

NSSL as a Secure FTP Proxy

NSSL can be run as a proxy process to front-end the NonStop FTPSERV or FTP process. With its SSL support, NSSL will enable secure communication to FTP clients or servers, which support FTP over SSL/TLS according to RFC-2228. SSL capable FTP clients are, for example, OutsideView, version 7.3 or above



NSSL as a SSL FTP proxy front-ending the standard NonStop FTPSERV process



Secure file transfer with SecurFTP running in client mode

Acting as a proxy server, NSSL will use secure FTP connections with the FTP partner and "tunnel" them to a plain FTP client or server.

The NSSL secure FTP proxy will intercept the communication on the FTP command socket to add encryption for both the command and data sockets. From the FTP server's or client's point of view the proxy acts as a normal FTP partner, while for the remote SSL FTP partner the NSSL proxy acts as a RFC-2228 compliant secure FTP server or client.

NSSL as a Plain FTP Client Proxy

NSSL can be used to overcome the limitation of the FTP client on the NonStop platform: Acting as a plain proxy for the FTP client, it can allow file transfers in PASSIVE mode initiated through the NonStop FTP client. The architecture is identical as with SecureFTP running in client mode with the only difference that the connections to the remote FTP server will not be encrypted.

NSSL as a plain FTP Server Proxy

NSSL can be used to monitor FTP traffic in which the NonStop system is the FTP server. NSSL will write a text file containing all commands issued by remote FTP users on the NonStop system allowing for a tight monitoring of FTP file transfers both in the Guardian and OSS File system. The architecture is identical as with SecureFTP running in server mode with the only difference that the connections to the remote FTP client will (optionally) not be encrypted.

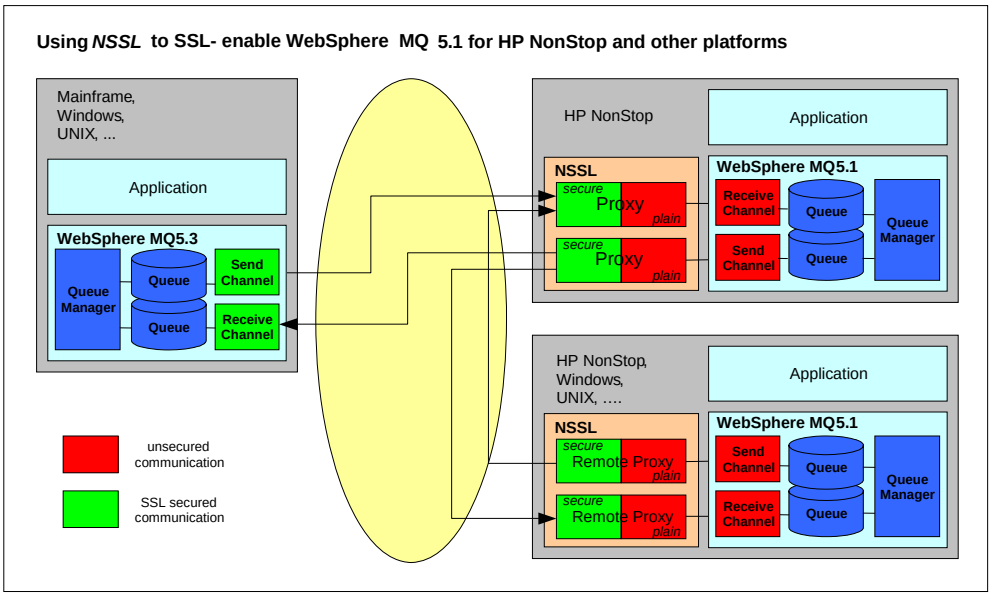
NSSL as a Secure ATTUNITY Server Proxy

NSSL can be run as a proxy process to front-end Attunity server components running on the NonStop server. Started as a secure Attunity server proxy,

NSSL is aware of the Attunity client/server protocol, to support the redirection of sessions to varying port numbers. Together with the Remote Proxy available from Crystal Point NSSL will enable secure communication with Attunity clients on remote systems.

NSSL as a Proxy to Secure IBM Websphere MQ

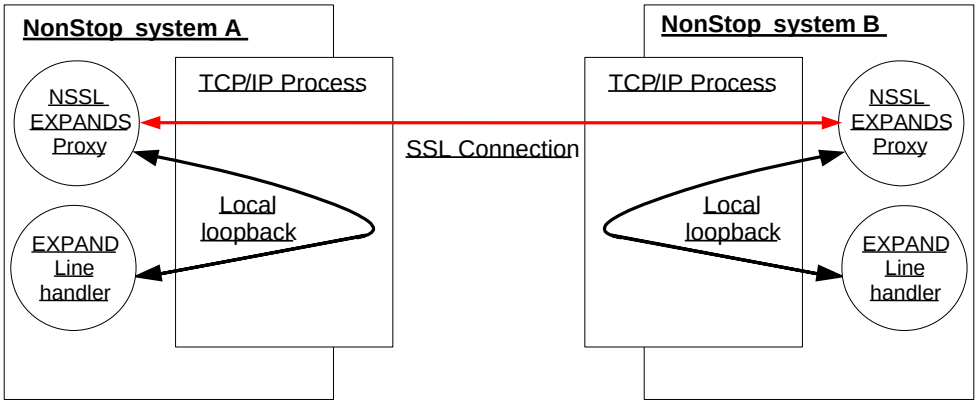
NSSL for MQ (MQS, MQC) enables SSL for the 5.1 release of WebSphere MQ running on a NonStop platform. NSSL will smoothly interact with the built-in SSL of WebSphere MQ 5.3 which is already available for other platforms. As a true any-to-any platform solution, NSSL will also encrypt WebSphere MQ traffic between multiple 5.1 nodes running on either the NonStop platform or other platforms.



NSSL as a proxy for WebSphere MQ

NSSL as a Proxy to Secure EXPAND Over IP Traffic

NSSL for EXPAND (EXPANDS run mode) encrypts EXPAND over IP traffic between two NonStop systems. It does so by creating a secure SSL session between the two systems as depicted in the following diagram:

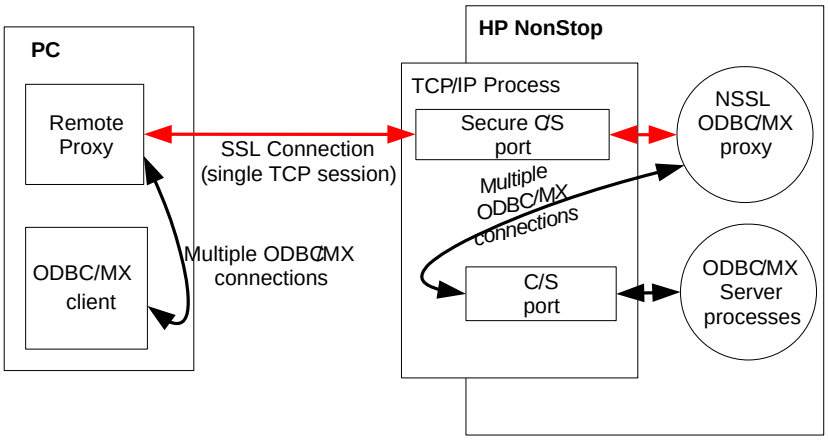


NSSL as a proxy for EXPAND over IP traffic

The EXPAND line handler will exchange UDP traffic with an instance of NSSL running on the same NonStop system; the two NSSL processes create an SSL TCP session between the two systems to forward the traffic.

NSSL as a Secure Proxy for ODBC/MX Traffic

NSSL for ODBC/MX traffic encrypts ODBC/MX traffic between client workstations and NonStop systems. ODBC/MX uses multiple TCP/IP sessions between a single client and the NonStop system, however in conjunction with the RemoteProxy available from Crystal Point, NSSL "tunnels" multiple sessions over a single one as shown in the following diagram:



NSSL as a proxy for ODBC/MX traffic

The "tunneling" approach has the following benefits:

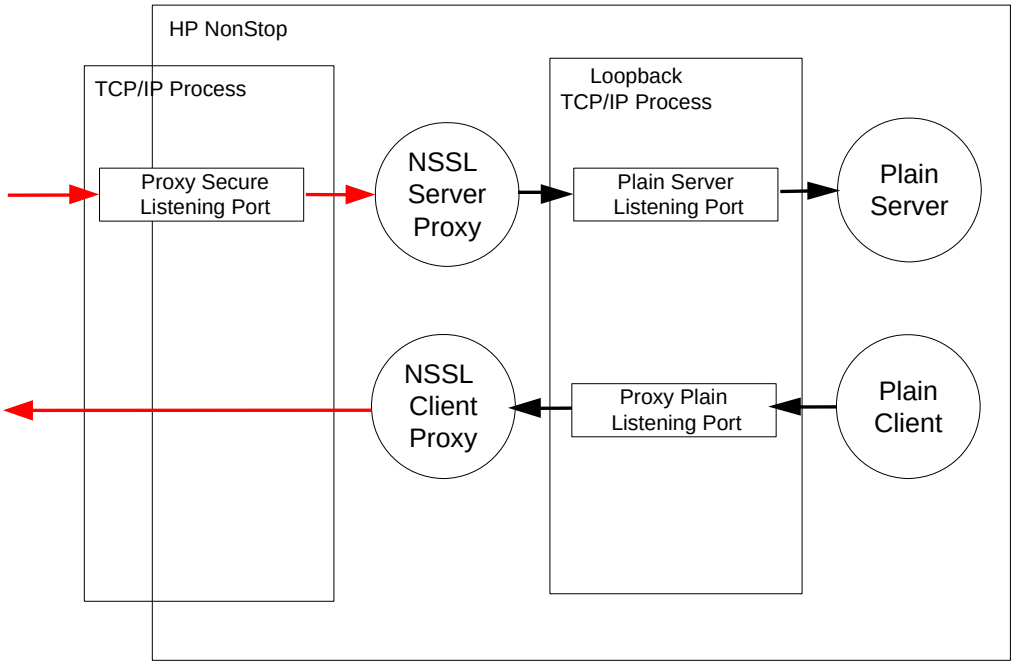
- It is firewall-friendly, as only a single port needs to be opened between the workstations and the clients.
- The configuration both of NSSL and the RemoteProxy is independent of the number of ports used by ODBC/MX.

Please see the NSSL Quickstart guide for a detailed description on how to set up the RemoteProxy component for ODBC/MX traffic.

Protecting Plain Ports with NSSL as a Multi-Homed Proxy

If NSSL is used with a proxy run mode, you can configure different TCP/IP process names for the listening and connecting sockets. One of the TCP/IP processes could even be a loopback-only process, without any connection to the network.

This "multi-homed" configuration allows protection of non-secure server ports from external access. It also prevents a client proxy from being hijacked by an external attacker.



NSSL as a multi-homed proxy with an internal loopback TCP/IP process

Using NSSL to Limit the Remote IP Addresses

NSSL can be configured to allow only certain remote IP addresses. By default, NSSL will allow connections from any IP address; this behaviour can be changed by

1. setting a "black list" of forbidden IP addresses using the DENYIP parameter.
2. setting a "white list" of allowed IP addresses using the ALLOWIP parameter.

Note: the black list will take precedence over the white list: if an IP address is matching both lists, it will NOT be allowed.

For details please see the description of the two parameters in the parameter reference.

Installation

System Requirements

To run NSSL in your environment your systems must meet the following requirements:

HP NonStop host:

- D.45 or later

For browser HTTP access:

- Any HTTP 1.0 compliant network browser

For browser (secure) HTTPS access:

- Any SSL 3.0 or TLS 1.0 compliant network browser with strong encryption (i.e. 128bit encryption)

For secure Telnet sessions:

- OutsideView 7.2 or later.
- or any SSL 3.0 or TLS 1.0 compliant telnet client

For secure FTP sessions:

- Any RFC-2228 compliant FTP client or server, such as OutsideView, version 7.3 or above.

For secure middleware transport sessions:

- The Remote Proxy available from Crystal Point.

For secure WebSphere MQ sessions:

- On HP NonStop Host: MQSeries for HP NonStop systems, Version 5 Release 1
- On the remote system:
    - either WebSphere MQ Version 5.3 (already SSL enabled)
    - or the Remote Proxy for MQS (included with NSSL/MQ) and a WebSphere MQ system

Installation on the NonStop Server

After you have downloaded the files to your workstation, you should transfer the NSSL installation archive (NSSLSINS.100) to your NonStop system, alter the file code and run the installation program.

Installing NSSL on the NonStop System

1. Using your favorite file transfer program, transfer the NSSL installation archive (NSSLSINS.100) in binary mode to your NonStop system. Copy the file to the subvolume you want to install NSSL on.
2. Alter the installation archive file code:

```
FUP ALTER NSSLSINS, CODE 100
```

3. Extract the archive by issuing the following command:

```
RUN NSSLSINS
```

The NSSL program files will now be copied to the current subvolume.

Considerations for installing on different NonStop server versions

NSSL is available for the following NonStop server versions:

- on K- or S-series systems, NSSL will run on any operating system equal or greater to Guardian release D45.
- NSSL is available for all Guardian releases on NonStop Integrity servers ("Itanium).

A full NSSL installation consists of the following object files:

- NSSL
- NSSLCOM
- SHOWLOG

NSSL and SHOWLOG are native object files and are delivered as two different object files for K/S-Series (type 700 file) or Itanium (type 800 file). NSSLCOM is an RISC object file and hence will run both on K/S series and Itanium.

The following table shows how the different versions of those three objects are delivered:

Component	Object Type	Naming
NSSL	native object	NSSL: 700 file NSSLI: 800 file
NSSLCOM	RISC object file	NSSLCOM: 100 file
SHOWLOG	native object	SHOWLOG: 700 file SHOWLOGI: 800 file

The rest of this manual assumes that the proper object files for your system have been renamed as follows:

- For K/S series systems no further action is necessary. The files NSSLI and SHOWLOGI can be deleted.
- For Itanium systems
    - the files NSSL and SHOWLOG should be deleted.
    - the file NSSLI should be renamed to NSSL
    - the file SHOWLOGI should be renamed to SHOWLOG

Installing the License File

For the run modes HTTPS, TELNETS, PROXYS, PROXYC, FTPS, FTPC, FTPCPLAIN, MQS and MQC you need a license file from Crystal Point which allows the usage of that run mode. The license file is tied to your system number.

The license file should be called LICENSE (default if not otherwise specified using the license parameter) and should reside on the same subvolume as the NSSL object file. If you need to put the license file in a different location you must use the PARAMETER LICENSE to specify the location. If there is a problem with the license file, NSSL will issue a message and terminate:

```
08:46:08.69|20|-----------------------------------------------------------
--------
08:46:08.69|10| comForte SWAP server version
T9999G06_18Sep2003_comForte_SSLD_S40_1031
08:46:08.69|10|using openssl version 0.9.7 - see http://www.openssl.org
08:46:08.70|10|config file:  '(none)'
08:46:08.70|10|runtime args: 'TELNETS;PORT 9023;TARGETPORT 65023'
08:46:08.70|20|--------- start settings for Logging  -----------
08:46:08.70|20| process name is $Y593
08:46:08.70|20| trace file is '*' ('*' means none)
08:46:08.71|20| max file length 20480000 bytes, length-check every 100
writes
08:46:08.71|20| console is '%' ('*' means none, '%' means home
terminal)
08:46:08.71|20| global maximum level is 9999, maximum dump length is
112
08:46:08.71|20|--------- end settings for Logging -------------
08:46:08.71|10|log level is 50
08:46:09.17|10|your system number is 12151
comForte NSSL server version T9999G06_18Sep2003_comForte_SSLD_S40_1031
--- Fatal Error:

 problem with checking license file:
 could not open license file 'LICENSE', error 4002

 --- aborting.
```

If the license file is proper you will see the expiration date in a log message during startup:

```
08:47:51.26|20|-------------------------------------------------------
--------
08:47:51.27|10| comForte SWAP server version T9999G06_18Sep2003_
comForte_SSLD_S40_1031
08:47:51.27|10|using openssl version 0.9.7 - see http://www.openssl.org
08:47:51.27|10|config file:  '(none)'
08:47:51.27|10|runtime args: 'TELNETS;PORT 9023;TARGETPORT 65023'
08:47:51.27|20|--------- start settings for Logging  -----------
08:47:51.28|20| process name is $Y594
08:47:51.28|20| trace file is '*' ('*' means none)
08:47:51.28|20| max file length 20480000 bytes, length-check every 100
writes
08:47:51.28|20| console is '%' ('*' means none, '%' means home
terminal)
08:47:51.28|20| global maximum level is 9999, maximum dump length is
112
08:47:51.29|20|--------- end settings for Logging -------------
08:47:51.29|10|log level is 50
08:47:51.62|10|your system number is 12151
08:47:51.81|10|license file check OK, license file 'LICENSE',
expiration is never
08:47:51.81|30|starting collecting of random data
08:47:54.92|10|collection of 64 bytes random data finished
08:47:55.22|20|dumping configuration:
[def  ] ALLOWIP              <*>
[def  ] CACERTS              <CACERT>
[def  ] CIPHERSUITES         <0.4,0.10,0.5>
[def  ] DENYIP               <>
[def  ] LICENSE              <LICENSE>
[def  ] LOGCONSOLE           <%>
[def  ] LOGFILE              <*>
[def  ] LOGFORMAT            <76>
[def  ] LOGLEVEL             <50>
[def  ] LOGMAXDUMP           <100>
[def  ] LOGMAXFILELENGTH     <20000>
[def  ] MAXVERSION           <3.1>
[def  ] MINVERSION           <3.0>
[run  ] PORT                 <9023>
[def  ] RANDOMFEED           <64>
[def  ] SERVCERT             <SERVCERT>
[def  ] SERVKEY              <SERVKEY>
[def  ] SERVKEYPASS          <??11??>
[def  ] SLOWDOWN             <0>
[def  ] SUBNET               <$ZTC0>
[def  ] TARGETHOST           <127.0.0.1>
[run  ] TARGETPORT           <65023>
[def  ] TARGETSUBNET         <$ZTC0>
[def  ] TESTWRONGDATASOCKET  <0>
08:47:55.31|50|OpenSSL cipherstring 'RC4-MD5:DES-CBC3-SHA:RC4-SHA:'
08:47:55.31|30|loading Server Certificate from file 'SERVCERT'
08:47:55.43|20|adding CA Certificate Chain Level 1/1: 'CACERT'
08:47:55.43|30|loading next Certificate Chain file from file 'CACERT'
08:47:55.53|20|Fingerprint of Root CA is
<F9E29DFC22D687C20C353BC2E37F959A>
08:47:55.53|30|loading private key from file 'SERVKEY'
08:47:55.61|10|DEFINE =TCPIP^PROCESS^NAME has value '\COMF.$ZTC0'
08:47:55.61|10|parameter SUBNET will be ignored
08:47:55.61|20|TCP/IP process is \COMF.$ZTC0
08:47:55.61|20|secure-to-plain proxy started on target host 127.0.0.1,
target port 65023, source port 9023
```

Running NSSL as a Plain HTTP Server

Since few configuration parameters are required for plain HTTP mode NSSL can be easily started as a web server by a single RUN command. After the NSSL process has been started you can immediately connect to your NonStop system with your browser.

To Start the NSSL Web Server

1. At the command prompt, issue the following command:

```
RUN NSSL/NAME $HTTP/ HTTP; SUBNET $ZTC0; PORT 8080
```

   where

   - the keyword "HTTP" designates the NSSL run mode as a HTTP server.
   - the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. You may omit this parameter, in which case NSSL will assume $ZTC0 as default.
   - the parameter "PORT" reflects the port number NSSL should listen on for HTTP connections. Note, that to start an NSSL web server on the well known HTTP port (80), SUPER group rights will be required.

2. NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

To Connect with a Browser

1. Point your browser to your NonStop system's IP address:

```
http://ipaddress:port
```

   where

   - *ipaddress* is the address of the TCP/IP process you started NSSL on
   - *port* the number you specified as PORT parameter in the RUN NSSL command. If you started NSSL to listen on the well known HTTP port (80) you can omit the ":*port*" portion.

2. Your browser should now display the NSSL welcome page.

Running NSSL as a Secure HTTPS Server

NSSL is delivered with a set of sample certificate and key files required for SSL. The default settings of the relevant SSL parameters point to these sample files to allow an out-of-the-box installation for evaluation purposes. Using the default settings, you can start NSSL as a secure web server by a simple RUN command.

Note: For a production installation, you should use your own server certificate. Please refer to "Configuring SSL for production" in the "SSL Reference" chapter for details.

To Start the NSSL Secure Web Server

1. At the command prompt, issue the following command:

```
RUN NSSL/NAME $HTTPS/ HTTPS; SUBNET $ZTC0; PORT 8443
```

   where

   - the keyword "HTTPS" designates the NSSL run mode as a secure web server.
   - the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. You may omit this parameter, in which case NSSL will assume $ZTC0 as default.
   - the parameter "PORT" reflects the port number NSSL should listen on for HTTPS connections. Note, that to start an NSSL secure web server on the well known HTTPS port (443), SUPER group rights will be required.

2. NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

To Connect with Your Browser

1. Point your browser to your NonStop system's IP address:

```
https://ipaddress:port
```

   where

   - *ipaddress* is the address of the TCP/IP process you started NSSL on
   - *port* the number you specified as PORT parameter in the RUN NSSL command. If you started NSSL to listen on the well known HTTPS port (443) you can omit the ":*port*" portion.

2. Your browser may now prompt you to accept the server certificate as it is not known to the browser. Choose to accept it.
3. Your browser should now display the NSSL welcome page indicating that it is operating in secure mode.

Note: NSSL is delivered with a test server certificate signed by a test CA which should only be used for testing. Your browser comes with a preloaded set of accepted certificates. Since "Test CA" is unknown to the browser, it will prompt the user to accept the connection. For a production installation, you should generate your own server certificate and make it acceptable for the browser, or apply for a server certificate at a certificate authority (CA) like Verisign or Thawte. See the "SSL Reference" chapter for details.

Running NSSL as a Secure Telnet Proxy

NSSL is delivered with an example configuration file (SSLCONF), which contains an SSL configuration for testing purposes. Using this configuration file, you can start NSSL as a secure Telnet proxy by a single RUN command.

Note: For a production installation, you should use your own server certificate. Please refer to "Configuring SSL for production" in the "SSL Reference" chapter for details.

To Start the NSSL Secure Telnet Proxy

1. Determine the Telnet server you want to install the secure proxy for and find out the TCP/IP process and port number it is listening on.
2. If the TELSERV process is running on TCP/IP process $ZTC0, port 23
   issue the following command at the command prompt:

```
RUN NSSL/NAME $STN0/ TELNETS; PORT 8423; CONFIG SSLCONF
```

   otherwise start the proxy with a command such as:

```
RUN NSSL/NAME $STN03/ TELNETS; SUBNET $ZTC03; PORT 8423;
  TARGETPORT 1023; CONFIG SSLCONF
```

   where

   - the keyword "TELNETS" designates the NSSL run mode as a secure telnet proxy.
   - the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. This should be the same process the TELSERV process runs on. As shown above, you may omit this parameter, in which case NSSL will assume $ZTC0 as default.
   - the parameter "PORT" reflects the port number NSSL should listen on for secure Telnet connections.
   - the parameter "TARGETPORT" reflects the port number of the TELSERV process the connections should be routed to. As shown above, you may omit this parameter, in which case NSSL will assume the well known telnet port 23 as default.

- the parameter "CONFIG" refers to the configuration file containing the example SSL configuration delivered with NSSL.

3 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Note: By specifying an additional parameter "TARGETHOST" and /or "TARGETSUBNET", you can also start a proxy on a TCP/IP process (or even system) other than that where the TELSERV listens on. However for security reasons, you should specify the "local loopback address" (127.0.0.1) as TARGETHOST. A local loopback avoids that unencrypted data traverses the network. In the above examples the TARGETHOST parameter has been omitted, since it will default to "127.0.0.1".

To Create a Secure Connection with Any Secure Telnet Client

Configure your SSL Telnet client to connect to the address and port number the NSSL secure telnet proxy listens for incoming connections. Make sure that the client has the SSL protocol enabled for the session.

To Create a Secure Telnet Connection with OutsideView

In the Session Settings dialog, I/O tab, select “Encrypt datastream using SSL” to enable encryption. Additional options allow definition of server certificate handling and cipher suites. Please refer to the documentation included with OutsideView for a full description of these options.

Running NSSL as a Secure Client/Server Proxy

Running NSSL as a Secure RSC Proxy

NSSL is delivered with an example configuration file (SSLCONF), which contains an SSL configuration for testing purposes. Using this configuration file, you can start NSSL as a secure client/server by a single RUN command. Remember: NSSL supports many connections as mentioned earlier. For the following example we used RSC. In most cases the difference is only in the Process which NSSL will transfer the data to.

Note: For a production installation, you should use your own server certificate. Please refer to "Configuring SSL for production" in the "SSL Reference" chapter for details.

To Start the NSSL Secure RSC Proxy

1. Determine the RSC Transaction Delivery Process (TDP) you want to install the secure proxy for and find out the TCP/IP process and port number it is listening on.
2. Assuming the TDP process is configured to running on TCP/IP process $ZTC0 and to listen on incoming port 12001, run NSSL as follows:

```
RUN NSSL/NAME $STN03/ PROXYS; SUBNET $ZTC03; PORT 8423;
  TARGETPORT 12001; CONFIG SSLCONF
```

   where

   - the keyword "PROXYS" designates the NSSL run mode as a secure client/server proxy.
   - the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. This should be the same process the TELSERV process runs on. As shown above, you may omit this parameter, in which case NSSL will assume $ZTC0 as default.
   - the parameter "PORT" reflects the port number NSSL should listen on for secure Telnet connections.
   - the parameter "TARGETPORT" reflects the port number of the TDP process the connections should be routed to.
   - the parameter "CONFIG" refers to the configuration file containing the example SSL configuration delivered with NSSL.

3. NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Note: By specifying an additional parameter "TARGETHOST" and / or "TARGETSUBNET", you can also start a proxy on a TCP/IP process (or even system) other than that where the TELSERV listens on. However for security reasons, you should specify the "local loopback address" (127.0.0.1) as TARGETHOST. A local loopback avoids that unencrypted data traverses the network. In the above examples the TARGETHOST parameter has been omitted, since it will default to "127.0.0.1".

Running NSSL as a Secure Attunity Proxy

NSSL is delivered with a set of sample certificate and key files required for SSL. The default settings of the relevant SSL parameters point to these sample files to allow an out-of-the-box installation for evaluation purposes. Using the default settings, you can start NSSL as a secure Attunity Proxy by a simple RUN command.

Note: For a production installation, you should use your own server certificate. Please refer to "Configuring SSL for production" in the "SSL Reference" chapter for details.

To Start the NSSL Secure Attunity Proxy

1. Check your Attunity configuration on the NonStop server, to determine the Attunity server you want to install the secure proxy for and find out the TCP/IP process and port number it is listening on.
2. Assuming the Attunity server process is configured to running on TCP/IP process $ZTC0 and to listen on incoming port 2551, run NSSL as follows:

```
RUN NSSL/NAME $STN03/ ATTUNITYS; SUBNET $ZTC0; PORT 8551;
  TARGETPORT 2551
```

where

- the keyword "ATTUNITYS" designates the NSSL run mode as a secure Attunity server proxy.
- the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. This should be the same process the Attunity server process runs on
- the parameter "PORT" reflects the port number NSSL should listen on for secure Attunity connections from clients.
- the parameter "TARGETPORT" reflects the port number of the Attunity server process the connections should be routed to.

3. NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Running NSSL as a Secure FTP Proxy

NSSL is delivered with a set of sample certificate and key files required for SSL. The default settings of the relevant SSL parameters point to these sample files to allow an out-of-the-box installation for evaluation purposes. Using the default settings, you can start NSSL as a secure FTP proxy by a simple RUN command.

To Start the NSSL FTP Server Proxy

1. At the command prompt, issue the following command:

```
RUN NSSL/NAME $FTPS/ FTPS; SUBNET $ZTC0; PORT 8421
```

where

- the keyword "FTPS" designates the NSSL run mode as a secure FTP proxy.
- the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. You may omit this parameter, in which case NSSL will assume $ZTC0 as default.
- the parameter "PORT" reflects the port number NSSL should listen on for FTP connections. Note, that to start an NSSL FTPS proxy server on a port number below 1024, SUPER group rights will be required.

2 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

To start the NSSL FTP Server Proxy with an Audit Log

1 Issue the following command at the command prompt:

```
RUN NSSL/NAME $FTPS/ FTPS; SUBNET $ZTC0; PORT 8421; AUDITFILE FTPAUDIT;
FTPALLOWPLAIN TRUE
```

where

- the keyword "FTPS", "SUBNET" and "PORT" reflects are identical than in the previous example.
- the parameter "AUDITFILE" specifies that an audit file containing commands from the remote FTP clients will be created.
- the parameter "FTPALLOWPLAIN" set to true means that FTP clients without encryption will be supported.

2 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

3 View the content of the file FTPAUDIT at any time to see the remote commands issued against the server proxy. You can view the content and dump it to an edit file using the SHOWLOG utility described in this manual. You can also use OSS tools such as grep or tail against the audit file.

To Start the NSSL FTP Client Proxy

1 Issue the following command at the command prompt:

```
RUN NSSL/NAME $FTPC/ FTPC; SUBNET $ZTC0, PORT 8021
```

where

- the keyword "FTPC" designates the NSSL run mode as a secure FTP client proxy.
- the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. This should be the same process which you will use for the FTP client. You may omit this parameter, in which case NSSL will assume $ZTC0 as default.
- the parameter "PORT" reflects the port number NSSL should listen on for connections from the NonStop FTP client.

2 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

To Start the NSSL FTP Client Proxy without Encryption

1 Issue the following command at the command prompt:

```
RUN NSSL/NAME $FTPC/ FTPCPLAIN; SUBNET $ZTC0, PORT 8021, PASSIVE 1
```

where

- the keyword "FTPCPLAIN" designates the NSSL run mode as a FTP client proxy without encryption.
- the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. This should be the same process which you will use for the FTP client. You may omit this parameter, in which case NSSL will assume $ZTC0 as default.
- the parameter "PORT" reflects the port number NSSL should listen on for connections from the NonStop FTP client.
- the parameter "PASSIVE" indicated that passive mode should be used for file transfer

2 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Running NSSL as a Secure WebSphere MQ Proxy

To Start the NSSL Secure WebSphere MQ Proxy Process for the Sending Channel

1 Determine the ip address or DNS name of the host on which the remote WebSphere MQ system runs. This information can be retrieved from the CONNAME parameter configured with the sending channel of the local WebSphere MQ instance.

2 Find out the port number on which the remote WebSphere MQ listens for it's receiving channel. Usually this is the registered port number 1414. See also the CONNAME param mentioned above.

3 Given that the CONNAME parameter of the local sending channel configuration is currently set to ip address 10.0.1.191 and port 1414 then the NSSL process is started using the following command:

```
RUN NSSL MQC; PORT 11414 ; TARGETHOST 10.0.1.191; TARGETPORT 1414
```

where

- the keyword "MQC" designates the NSSL run mode as a secure WebSphere MQ proxy forwarding local sending channels to remote WebSphere MQ instances.
- the parameter "PORT" reflects the port number NSSL should listen on for the connection from the local WebSphere MQ sending channel.

- the parameter "TARGETHOST" reflects the ip address of the host (for example 10.0.1.191) on which the remote WebSphere MQ runs.
- the parameter "TARGETPORT" reflects the port number on which the remote WebSphere MQ listens on for it's receiving channel.

To Start the NSSL Secure WebSphere MQ Proxy Process for the Receiving Channel

1. Determine the port number on which the local WebSphere MQ listens for it's receiving channel. This is usually the registered port 1414.
2. The NSSL process is started using the following command:

```
RUN NSSL MQS; PORT 1414 ; TARGETHOST 127.0.0.1; TARGETPORT 21414
```

where

- the keyword "MQS" designates the NSSL run mode as a secure WebSphere MQ proxy forwarding inbound receiving channels to local WebSphere MQ instances.
- the parameter "PORT" reflects the port number NSSL should listen on for inbound receiving channels.
- the parameter "TARGETHOST" reflects the ip address of the local WebSphere MQ system. In this case it is assumed that the NSSL process runs on the same host as the local WebSphere MQ system. Therefore, the loop back address 127.0.0.1 is used.
- the parameter "TARGETPORT" reflects the port number on which the local WebSphere MQ listens for it's receiving channel. Please note, that port 1414 is already occupied by the NSSL as the listener port. Therefore, the local WebSphere MQ must be configured to listen on some other port, in our example on 21414.

Running NSSL as an SSL Tunnel for EXPAND-Over-IP Lines

Starting NSSL

Creating an SSL tunnel for an EXPAND-over-IP line requires running an NSSL process in EXPANDS mode for the line handler on both sides of the connection. The configuration of the NSSL processes can be easily derived from the existing line handler configuration of EXPAND-over-IP line. To enable the tunneling, only a single line handler attribute needs to be changed.

The following steps explain how install an SSL tunnel process for an EXPAND-over-IP line handler. Please note that these steps need to be performed on both NonStop servers connected by the line.

To Start the NSSL EXPANDS Tunnel

1. For the EXPAND-over-IP line handler you want to secure, determine the values of the following configuration attributes using the SCF INFO LINE DETAIL command:
   - ASSOCIATEDEV
   - DESTIPADDR
   - DESTIPPORT
   - SRCIPADDR
   - SRCIPPORT
2. Start the NSSL EXPANDS process as follows:

```
RUN NSSL EXPANDS; SUBNET <associatedev>;
  DESTIPADDR <destipaddr>; DESTIPPORT <destipport>;
  SRCIPADDR  <srcipaddr>;  SRCIPPORT  <srcipport>
```

   where you fill in the respective values you determined in step 1.

3. NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Note 1: For a production installation it is recommended to run NSSL EXPANDS as a generic process sending log output to a file.
Note 2: An NSSL process has to be started on each system, please see the diagram in "NSSL as a Proxy to Secure EXPAND Over IP Traffic" for details.

To Activate the SSL Tunnel for the EXPAND Line

1. Using SCF, alter the configuration of the EXPAND line as follows:

```
> ASSUME LINE <line>
> ABORT
> ALTER, DESTIPADDR 127.0.0.1
> START
```

2. After the tunnel is properly configured on both sides, you should see NSSL log messages similar to the following:

```
$EXPS |27Apr05 12:31:41.01|50|E1| tunnel connect succeeded, tunnel
ready
```

   or

```
$EXPS |27Apr05 12:37:26.78|50|E1| accepted tunnel connection, tunnel
ready
```

The EXPAND line should then show the "READY" state.

Note: Again, that change in the SCF configuration has to be done on both systems.

Load Balancing and Fault-Tolerance of EXPAND over SSL

Using the EXPAND multi-line or multi-CPU path feature, it is possible to distribute the CPU load generated by the SSL encryption of the EXPAND traffic across multiple CPUs. Having multiple EXPAND SSL lines connecting systems will also provide fault-tolerance against CPU and other failures. If an EXPAND line goes down due to an NSSL EXPANDS process terminating for any reason, the traffic will be redirected over the remaining lines.

EXPAND Multi-Line versus Multi-CPU Paths

The choice between Multi-Line or Multi-CPU paths (SUPERPATH) is influenced by the nature of the traffic between the systems, as well as the load-balancing and fault-tolerance goals to be achieved.

Multi-Line and Multi-CPU paths over SSL differ in the following aspects:

- CPU consumption
  Since Multi-CPU paths have a separate LH process for each line, the NSSL processes can be configured to use the same CPU, reducing message-system hops between CPUs for the Loopback communication. Measurements have shown a lower CPU consumption of about 250ms/MB transferred data on an S72000 system.
- Load-balancing
  Multi-CPU path will assign a particular line to any pair of communicating processes. Hence, if a single pair of communicating process is generating a high traffic load, such as a FUP DUP or an RDF Extractor/Replicator, this traffic will burden a single CPU.
  Multi-Line paths will distribute traffic evenly across all available lines, independently of the number and CPUs of the processes commicating over EXPAND. Load will also be re-distributed dynamically and transparently, if a CPU of an NSSL EXPANDS process is heavily loaded by processes with a higher priority. Hence, bandwidth can be preserved, even if the NSSL processes run at a low priority to avoid impact on critical application processes.
- Fault-Tolerance
  With Multi-CPU paths a single line is assigned to a communication link between a requestor and a server. If this line goes down, a communication error will be reported to the requestor, and the communication link will have to be re-established.
  A failure of a single with a Multi-line path will be completely transparent to the application and the traffic will be re-routed automatically.
- Throughput
  The highest maximum thougput can be achieved with Multi-CPU paths. Measurements showed a throughput of up to 1-5 MB/s per CPU for FESA/100Mbit connected systems, with a linear scalability for multiple requestor/server pairs running in different CPUs (e.g. 6MB/s 1with 4 pairs).
  Multi-line paths have a lesser maximum throughput, as all traffic is handled by a single LH process. Measurements have shown a throughput of 1-4 MB/s for FESA/100Mbit connected systems with a single requestor server pair and a total maximum throughput of about 3 MB/s with multiple pairs.

Optimizing Throughput

The following configuration properties & setup can impact the overall throughput over an EXPAND over SSL path:

- LIF DataForwardCount (DFC) and DataForwardTime (DFT)

  Reducing the values DFC and DFT can increase the throughput for an EXPAND over SSL line. On S86000 or NS systems, DFC can be set to 1, reducing response time to a minimum. On slower systems, DFT/DFC should be set to the smallest possible value (e.g. DFT=1ms, DFC=10)

- CPU selection of NSSL EXPANDS processes with multi-line paths

  Starting an NSSL EXPANDS line process in primary CPU of a EXPAND line handler process handling multi-line path can severely decrease the overall throughput. For an optimal performance even in case of a takeover of the lime handler backup, it is recommended to run the NSSL EXPANDS processes in CPUs not used by the LH process.

Multi-Line Path Installation Sample

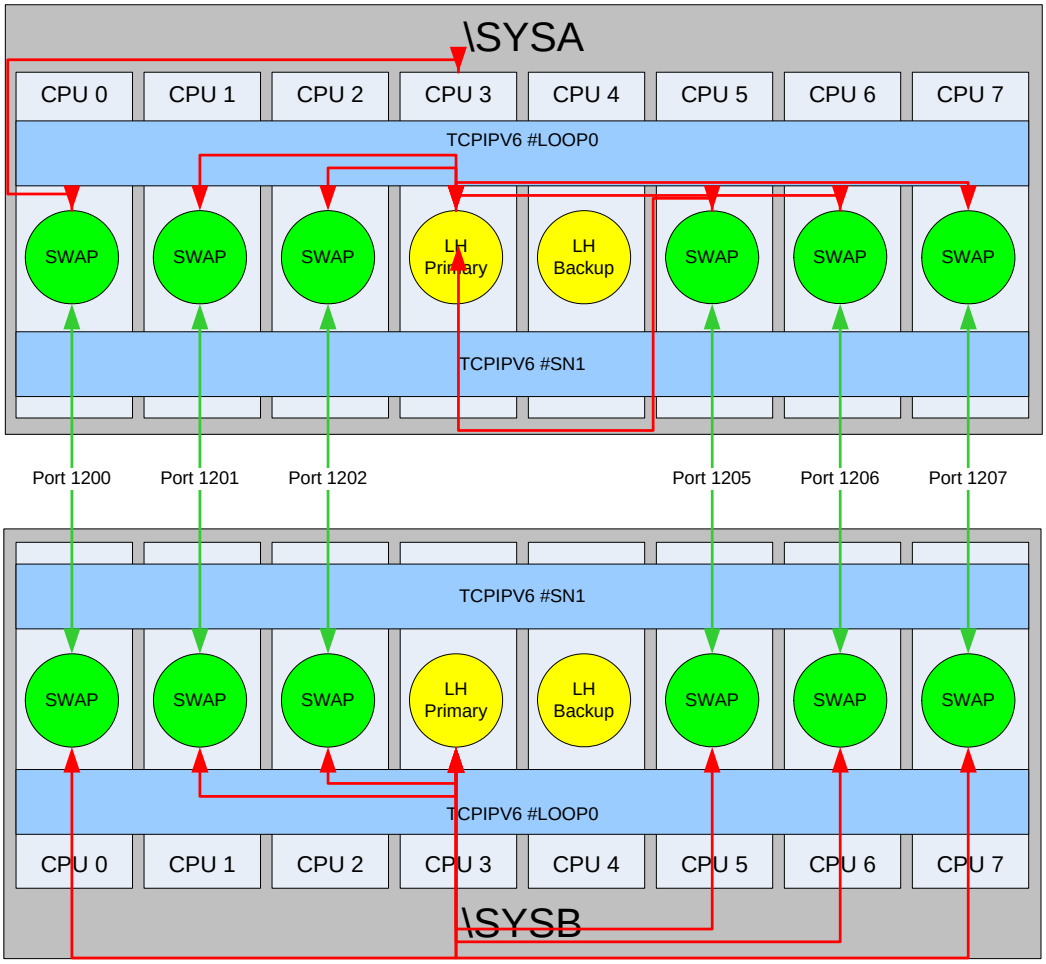
The following sample configuration illustrates how to optimize throughput, distribute CPU load and achieve fault-tolerance.

Assumptions

- \SYSA and \SYSB to be connected over EXPAND SSL
- Systems have 8 CPUs each
- TCPIPv6

Configuration

The following figure shows a complete setup:



The following steps have been performed for the above setup:

1. An Expand Multi-Line path was created on each system
   - 2 CPUs were selected for the LH primary and backup
   - To distribute SSL CPU load over the remaining CPUs, 6 lines were created for the path
   - A unique port number was selected for each line (SRCPORT and DESTPORT can be identical)
   - DESTIPADDR of all lines was set to the loopback address (127.0.0.1)
2. 6 NSSL EXPANDS persistent processes were created on both systems 0
   - A different CPU was selected for each SSL process
   - The SSL tunnel was associated to the line using the same SRCPORT and DESTPORT parameters as in the line configuration.
   - The DESTIPADDR parameter of the NSSL EXPANDS processes were set to the remote system's IP address.

Running NSSL as an SSL Tunnel for ODBC/MX Connections

Note: The configuration of NSSL for ODBC/MX differs from the configuration for ODBC/MP. This section describes the configuration of NSSL for ODBC/MX, please see the section "NSSL as a Secure Proxy for Generic TCP/IP Client Server Applications" for the configuration of NSSL for ODBC/MP.

To encrypt ODBC/MX traffic, NSSL needs to be configured in conjunction with the RemoteProxy component running on the workstation where the ODBC/MX client resides. Please see the NSSL Quickstart guide for a detailed description on how to set up the RemoteProxy component for ODBC/MX traffic.

Note: For a production installation, you should use your own server certificate. Please refer to "Configuring SSL for production" in the "SSL Reference" chapter for details.

To Start the NSSL Secure ODBC/MX Proxy

1. Run NSSL as follows:

```
RUN NSSL/NAME $SODBC/ ODBCMXS; SUBNET $ZTC03; PORT 8888; CONFIG SSLCONF
```

   where

   - the keyword "ODBCMXS" designates the NSSL run mode as a secure ODBC/MX proxy.
   - the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. This should be the same process the MXAOAS process runs on.
   - the parameter "PORT" reflects the port number NSSL should listen on for secure incoming connections.
   - you must not specify a value for TARGETPORT.
   - the parameter "CONFIG" refers to the configuration file containing the example SSL configuration delivered with NSSL.

2. NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Note: By specifying an additional parameter "TARGETHOST" and / or "TARGETSUBNET", you can also start a proxy on a TCP/IP process (or even system) other than that where the TELSERV listens on. However for security reasons, you should specify the "local loopback address" (127.0.0.1) as TARGETHOST. A local loopback avoids that unencrypted data traverses the network. In the above examples the TARGETHOST parameter has been omitted, since it will default to "127.0.0.1".

Configuring and Running NSSL

Configuration Overview

NSSL processes can be flexibly configured by a set of configuration parameters which can be specified by the following means:

- A configuration file
- PARAM commands
- Startup command line parameters

The different options to specify a configuration for NSSL allow system administrators to easily manage installations with multiple NSSL processes running on multiple TCP/IP processes and ports as well as in different modes. For example, multiple NSSL secure proxy processes with a an identical SSL configuration can share the same configuration file, while process-unique parameters such as proxy port, target host and port can be specified on the command line.

On startup, NSSL parses the given configuration parameters sources. A single parameter may be specified in multiple sources, e.g. in the configuration file and on the startup command line. In this case, NSSL will process parameters with the following precedence (highest to lowest):

1. PARAM parameter
2. Configuration file parameter
3. Startup line parameter

This means that parameter given in the configuration file will override the value given for the same parameter on the startup line. Likewise, a parameter value given as PARAM command will override any value specified in the configuration file.

All NSSL parameters can be specified in any of the configuration parameter sources, with the following exceptions:

- the run mode of an NSSL process is specified explicitly on the command line as first startup line parameter. This parameter defines if NSSL acts as a HTTP (web) server, a HTTPS (secure web) server and so forth for all supported modes (see "Starting NSSL" for a complete list).
- the configuration file to be used as a parameter source can only be specified as a PARAM or startup line parameter, as it is meaningless in a configuration file itself.

Regardless which way they are specified, parameter names are case insensitive.

The Configuration File

The configuration file is an edit type file which can be adapted with a standard NonStop editor such as TEDIT. The name of the file that an NSSL process should use as configuration source is passed to the program during startup (see "Starting NSSL" for details).

The file contains entries of the form

```
parameter-name  parameter-value
```

Like in the standard TCP/IP configuration files, any lines starting with a "#" character are interpreted as comments. The following printout is the contents of the sample configuration file for running NSSL as a secure web server:

```
# sample configuration file for an NSSL secure web server

#-------------------------------------------------------------------------
-----------
# general settings

# TCP/IP process the web server runs on
SUBNET       $ZTC0

# HTTPS server port where NSSL listens for incoming SSL browser
connections
# we use the well known HTTPS port (requires SUPER access to start
NSSL)
PORT        443

# subvolume the web server returns requested resources from
HTTPBASE      $DATA1.COMFHTML
# we also use a zip archive filled with HTTP contents
HTTPZIP      $DATA1.NSSL.HTTPZIP

#-------------------------------------------------------------------------
-----------
# log configuration
# set the level
LOGLEVEL 50
# enable console logging to $0
LOGCONSOLE $0
# additionally log to file
LOGFILE $DATA1.NSSL.HTTPSLOG

#-------------------------------------------------------------------------
-----------
# SSL configuration
# our server certificate and private key
SERVCERT      $DATA1.COMFNSSL.MYCERT
SERVKEY       $DATA1.COMFNSSL.PRIVKEY
SERVKEYPASS   myprivatepassword
# our server cert was issued by verisign
CACERTS       $DATA1.COMFNSSL.VERISIGN
# we only accept the strongest cipher suite with 3DES
CIPHERSIUTES    0.10
```

PARAM Commands

NSSL configuration parameters can be specified as PARAM commands as follows:

```
PARAM <parameter name> <parameter value>
```

All available NSSL parameters can be specified as PARAM commands.

The following example demonstrates how to start an NSSL plain HTTP server listening on $ZTC03, port 8080, using PARAM commands:

```
> PARAM PORT 8080
> PARAM SUBNET $ZTC03
> PARAM HTTPBASE COMFHTML
> RUN NSSL/ NAME $HTTP, NOWAIT/ HTTP
```

Startup Line Parameters

NSSL configuration parameters can be passed on the startup line as follows (for a complete description of the RUN NSSL command see section "Starting NSSL"):

```
<parameter name> <parameter value>; <parameter name> <parameter value>; ...
```

The following example demonstrates how to start a multiple NSSL secure proxies sharing the same SSLCONF configuration file:

```
> PARAM CONFIG SSLCONF
> RUN NSSL /NAME $STN0, CPU 0, NOWAIT/ TELNETS; SUBNET $ZTC0; PORT 8023
> RUN NSSL /NAME $STN1, CPU 1, NOWAIT/ TELNETS; SUBNET $ZTC1; PORT 8023
> RUN NSSL /NAME $STN2, CPU 2, NOWAIT/ TELNETS; SUBNET $ZTC2; PORT 8023
> RUN NSSL /NAME $STN3, CPU 3, NOWAIT/ TELNETS; SUBNET $ZTC3; PORT 8023
```

Starting NSSL

You create an NSSL process by issuing a TACL RUN command using the following syntax:

```
RUN NSSL / runoptions / mode [ ; paramname paramvalue; ... ]
```

where

- *runoptions* are the standard Guardian RUN options, such as IN, CPU or TERM
- *mode* defines the run mode of the NSSL process with the following valid keywords:

FTPC	FTP client proxy
FTPS	FTP server proxy
FTPCPLAIN	FTP client proxy without encryption
HTTP	HTTP server
HTTPS	Secure HTTP server
PROXYS	Generic SSL server proxy
PROXYC	Generic SSL client proxy
TELNETS	Secure Telnet proxy
ATTUNITYS	Secure Attunity proxy
MQS	Secure WebSphere MQ server proxy
MQC	Secure WebSphere MQ client proxy
EXPANDS	Secure EXPAND over IP proxy
ODBCMXS	Secure Proxy for ODBC/MX

- *paramname paramvalue*; ...
  is a list of NSSL configuration parameter settings as described in the previous section.

Note: When you start NSSL in NOWAIT mode, make sure you have disabled logging to the home terminal. To do so, set the following PARAM: PARAM LOGCONSOLE *

Security Considerations

While SSL is a very powerful and flexible protocol to encrypt TCP/IP traffic, it has to be used properly to be protected against some common attacks. The two most important factors in making an SSL installation fully secure are:

- protecting against the man-in-the middle attack through proper usage of certificates
- protecting the private key file

Note: Ignoring those two factors will result in a system open to well-known attacks. Please read this section and follow the recommendations to make sure you are deploying SSL properly.

Protecting Against the Man-in-the-Middle Attack

The man-in-the-middle attack is based on a weakness of the TCP/IP protocol which allows adding an "intermediary" between two systems communicating via TCP/IP.

To protect against that kind of attack, SSL uses certificates. See the following sections of this manual for more information:

- Chapter SSL Reference, section X.509 Certificates
- Chapter SSL Reference, section Configuring SSL for Production

Make sure to generate your own certificates for production and to configure all your SSL clients to verify the certificates used by the SSL server.

Protecting the Private Key File

If an attacker gets access to the private key file, he can attack the SSL protocol in various ways. Therefore it is important that you protect the private key file residing on your NonStop system.

The private key file is created during the generation of your certificates and is a file in your Guardian file system. The location of the file is configured using the parameter SERVKEY. Standard procedures (such as Safeguard ACL's) should be employed so that only the NSSL process can open that file.

The private key file is encrypted using a so-called pass phrase. An attacker needs both the private key file and the pass phrase for a successful attack. The pass phrase is configured through the SERVKEYPASS parameter, that parameter is probably present in some startup file or macro. This startup file needs again to be protected properly.

Note: Never send the private key file and/or the pass phrase to anybody via e-mail. Make sure the file resides only on your NonStop system and is properly protected via Safeguard.

If the Private Key Is Compromised

If you have reason to believe that your server private key file has been compromised, you should immediately install a new server certificate along

with a private key file encrypted with a different pass phrase. Using the Certificate Tools, you should generate a new Certificate Signing Request (CSR). If you obtained the server certificate from a third party or corporate CA, you should then send the CSR to the CA. If you are acting as your own CA, you should use your CA certificate and private key to issue a new server certificate.

Note: If you authenticate the NSSL server in your clients, you should consider to base trust on the Root CA certificate (e.g. check the Root CA fingerprint). In case the server certificate is compromised you can simply replace it without having to update your client configuration.

NSSL Parameter Reference

This section describes all available NSSL parameters in alphabetical order. Note, that parameter names are case insensitive independently of the source.

Parameter Overview

The following table lists all available NSSL parameters and their meanings:

Parameter	Meaning
ALLOWCERTERRORS	allows selective overriding of certificate validation errors.
ALLOWIP	limits allowed remote IP addresses.
AUDITASCIIONLY AUDITASCIIDUMPLENIN AUDITASCIIDUMPLENOUT AUDITCONSOLE AUDITLEVEL AUDITFILE AUDITFILERETENTION AUDITFORMAT AUDITMAXFILELENGTH	control the creation of an audit file containing the remote FTP commands in run mode FTPS or the socket activities in run modes PROXYS,PROXYC,PROXY,MQS,MQC,ODBCMXS.
CACERTS	file names of a DER encoded X.509 CA certificates representing a certificate chain signing the certificate configured with the CLIENTCERT or SERVCERT parameter.
CIPHERSUITES	list of cipher suites that will be accepted by a secure NSSL process
CLIENTAUTH	Enforced client authentication when running as SSL server: a certificate signing the certificates the client is using for SSL client authentication. [That parameter was called TRUST in versions prior to 1043, for details see "Trust". (NSSL Parameter Reference)]
CLIENTCERT	file name of a DER encoded X.509 client certificate.
CLIENTKEY	the private key to be used for the client certificate.
CLIENTKEYPASS	password for reading the (encrypted) private key file.
CONFIG	file name of an NSSL configuration file.

Parameter	Meaning
CONFIG2	allows the usage of a second configuration file with different security settings
CONTENTFILTER	Activates content-filtering in run modes PROXY, PROXYS and PROXYC
DENYIP	limits allowed remote IP addresses.
DONOTWARNONERROR	log selected errors with LOGLEVEL 20 rather than as WARNING
FTPALLOWPLAIN	allows plain FTP traffic when NSSL is run in FTPS mode
FTPCALLOW200REPLY	sets compatibility for older FTP/TLS servers when run in FTPC mode
FTPLOCALDATAPORT	controls the value of the local port on the NonStop system of the data connection in FTPC mode with PASSIVE set to TRUE.
FTPMAXPORT	the maximum port number NSSL will use for FTP data connections.
FTPMINPORT	the minimum port number NSSL will use for FTP data connections.
HTTPBASE	the location (subvolume) an NSSL HTTP server returns requested resources from.
HTTPZIP	specifies a ZIP file an NSSL HTTP server should return requested resources from.
INTERFACE	controls the IP address NSSL will bind to for connections made to NSSL.
KEEPALIVE	specifies if keep alive messages are sent to TCP/IP sockets.
LICENSE	the location for the license file of NSSL.
LOGCONSOLE	determines if log messages are written to a console.
LOGEMS	determines if log messages are written to EMS.
LOGFILE	determines if log messages are written to a file.
LOGFILERETENTION	controls the number of log files kept after rollover occurs
LOGFORMAT	controls the format of the log messages that are written to the console or log file.
LOGFORMATCONSOLE	controls the format of the log messages that are written to the console.
LOGFORMATEMS	controls the format of the log messages that are written to EMS.
LOGFORMATFILE	controls the format of the log messages that are written to a log file.
LOGLEVEL	determines which messages will be written the log file.
LOGLEVELCONSOLE	allows to set different log level for LOGCONSOLE output
LOGLEVELEMS	allows to set different log level for LOGEMS output
LOGLEVELFILE	allows to set different log level for LOGFILE output
LOGMEMORY	Allows regular logging of NSSL's memory usage to the log output
LOGMAXFILELENGTH	controls the maximum size of the log file.
MAXSESSIONS	Limits the number of parallel connections in run modes PROXYS, PROXYC, PROXY, TELNETS
MAXVERSION	maximum admissible SSL/TLS protocol version.
MINVERSION	minimum admissible SSL/TLS protocol version.

Parameter	Meaning
PASSIVE	sets direction of control socket connection in FTPC mode.
PEERCERTCOMMONNAME	for verification of remote certificates.
PEERCERTFINGERPRINT	for verification of remote certificates.
PORT	the port the NSSL server listens on for incoming connections.
PTCPIPFILTERKEY	Sets the filter key to enable round robin filtering
SERVCERT	file name of a DER encoded X.509 server certificate.
SERVKEY	the private key to be used for the server certificate.
SERVKEYPASS	password for reading the (encrypted) private key file.
SLOWDOWN	adds delay to processing resulting in slower encryption/decryption with less CPU usage
SOCKSHOST SOCKSPORT SOCKSUSER	Configures NSSL as SOCKS Version 4 client in run modes FTPC, FTPCPLAIN, PROXY or PROXYC
SUBNET	the name of the TCP/IP process NSSL should listen on for connections.
SWAPCOMSECURITY	restricts execution of NSSLCOM commands.
TARGETHOST	the IP address an NSSL proxy should route connections to.
TARGETINTERFACE	controls the IP address NSSL binds to for outgoing connections.
TARGETPORT	the port number an NSSL proxy should route connections to.
TARGETSUBNET	the name of the TCP/IP process NSSL should use for outgoing connections.
TCPIPHOSTFILE	Sets the DEFINE = TCPIP^HOST^FILE
TCPIPNODEFILE	Sets the DEFINE = TCPIP^NODE^FILE
TCPIPRESOLVERNAME	Sets the DEFINE = TCPIP^RESOLVER^NAME
TCPNODELAY	Activates RFC1323 on all sockets
TRUST	when running as SSL client: list of fingerprints of trusted CA or server certificates. [In versions prior to 1043 that parameter enforced SSL client authentication; this is now done with the new parameter CLIENTAUTH. For details see "CLIENTAUTH" in the "NSSL Parameter Reference".]

ALLOWCERTERRORS

Use this parameter to allow selective overriding of certificate validation errors.

Parameter Syntax

```
ALLOWCERTERROS number1 [, number2, ...]
```

Arguments

number

comma-separated lists of certificate errors which NSSL should ignore. The error numbers are defined in the openssl sources uses for NSSL (see Considerations).

Considerations

- **Note**: the usage of this parameter may compromise the security of your configuration. Use only as workaround and with care.
- The paramter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section "Command Interface NSSLCOM" for details.

The following table lists error numbers and names as defined in the openssl sources:

Error Name	Error number
X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT	2
X509_V_ERR_UNABLE_TO_GET_CRL	3
X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE	4
X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE	5
X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY	6
X509_V_ERR_CERT_SIGNATURE_FAILURE	7
X509_V_ERR_CRL_SIGNATURE_FAILURE	8
X509_V_ERR_CERT_NOT_YET_VALID	9
X509_V_ERR_CERT_HAS_EXPIRED	10
X509_V_ERR_CRL_NOT_YET_VALID	11
X509_V_ERR_CRL_HAS_EXPIRED	12
X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD	13
X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD	14
X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD	15
X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD	16
X509_V_ERR_OUT_OF_MEM	17
X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT	18
X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN	19
X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY	20
X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE	21
X509_V_ERR_CERT_CHAIN_TOO_LONG	22
X509_V_ERR_CERT_REVOKED	23
X509_V_ERR_INVALID_CA	24
X509_V_ERR_PATH_LENGTH_EXCEEDED	25
X509_V_ERR_INVALID_PURPOSE	26
X509_V_ERR_CERT_UNTRUSTED	27
X509_V_ERR_CERT_REJECTED	28
X509_V_ERR_SUBJECT_ISSUER_MISMATCH	29
X509_V_ERR_AKID_SKID_MISMATCH	30
X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH	31
X509_V_ERR_KEYUSAGE_NO_CERTSIGN	32
X509_V_ERR_UNABLE_TO_GET_CRL_ISSUER	33
X509_V_ERR_UNHANDLED_CRITICAL_EXTENSION	34

Error Name	Error number
X509_V_ERR_KEYUSAGE_NO_CRL_SIGN	35
X509_V_ERR_UNHANDLED_CRITICAL_CRL_EXTENSION	36
X509_V_ERR_APPLICATION_VERIFICATION	50

Default

If omitted, NSSL will work normally (all certificate validation errors are treated as such and connection attempts will fail)

Example

```
ALLOWCERTERRORS 10
```

This will temporarily allow expired certificates.

ALLOWIP

Use this parameter to specify which remote IP addresses are to be allowed to establish sessions ("white list").

Parameter Syntax

```
ALLOWIP range
```

Arguments

range

specifies a set of valid remote IP addresses. Valid values are

- * for all IP addresses
- single IP addresses such as 10.1.1.11
- a subnet such as 10.1.1.*
- comma-separated lists of single IP addresses and subnets

Considerations

- See section "Using NSSL to limit the remote IP addresses" for the concept of remote IP filtering.
- The parameter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.
- The parameter can be preceded by a single character modifying the directions in which the filter is applied:
  - none or B: both directions (default)
  - A: only applied to accepting socket
  - C: only applied to connecting socket

Default

If omitted, NSSL will use * to allow all remote IP addresses

Example

```
ALLOWIP 10.0.1.*, 10.0.2.*
```

AUDITASCIIONLY

Use this parameter to define how NSSL writes raw data to the audit log.

Parameter Syntax

```
AUDITASCIIONLY TRUE | FALSE
```

Arguments

TRUE

Data will be dumped in ASCII format, binary values will be represented as <hh> where hh is the hex representation

FALSE

Data will be dumped as full hex dump. This consumes a lot of resources but provides the most complete view

Default

By default, a value of TRUE will be used

Considerations

Audit messages will depend on the run mode – see parameter AUDITLEVEL for details

See also parameters AUDITASCIIDUMPLENIN and AUDITASCIIDUMPLENOUT

AUDITASCIIDUMPLENIN

Use this parameter to define how many bytes of incoming messages are written to the audit log when AUDITASCIIONLY is set to TRUE.

Parameter Syntax

```
AUDITASCIIDUMPLENIN number
```

Arguments

`-1`

means that each incoming message will be dumped fully

`N`

means that only the first N bytes of each incoming message will be dumped

Default

By default, a value of -1 will be used

Considerations

See parameter AUDITASCIIONLY

AUDITASCIIDUMPLENOUT

Use this parameter to define how many bytes of outgoing messages are written to the audit log when AUDITASCIIONLY is set to TRUE.

Parameter Syntax

```
AUDITASCIIDUMPLENIN number
```

Arguments

`-1`

means that each outgoing message will be dumped fully

`N`

means that only the first N bytes of each outgoing message will be dumped

Default

By default, a value of -1 will be used

Considerations

See parameter AUDITASCIIONLY

AUDITCONSOLE

Use this parameter to define if and to what console device NSSL audit messages are written to.

Parameter Syntax

`AUDITCONSOLE * | % | $0 | auditdevice`

Arguments

`*`

means that no audit messages are written to a console

`%`

means that audit messages are written to the home terminal of the NSSL process

`$0`

audit messages are written to $0

`auditdevice`

audit messages are written the given device (e.g. $DEV.#SUBDEV)

Considerations

- Audit messages will depend on the run mode – see parameter AUDITLEVEL for details.

Default

By default, audit messages will be not be written to a device ("*")

See also:

AUDITFILE, AUDITLEVEL, AUDITFORMAT

AUDITFILE

Use this parameter to define if and to what file NSSL audit messages are written to.

Parameter Syntax

```
AUDITFILE * | file
```

Arguments

`*`

means that audit messages are written to a file

file

the name of the auditfile

Default

By default, no audit messages are written to a file ("*")

Considerations

- Audit messages will depend on the run mode – see parameter AUDITLEVEL for details

See also:

AUDITCONSOLE, AUDITLEVEL, AUDITFORMAT

AUDITFILERETENTION

Use this parameter to control how many audit files NSSL keeps when audit file rollover occurs.

Parameter Syntax

```
AUDITFILERETENTION n
```

Arguments

`n`

number of audit files to keep

Default

By default, 10 files are kept.

Considerations

- a minimum of 10 is enforced for that parameter
- See section “Logfile/Auditfile rollover using round robin” for details on logfile rollover.

See also:

AUDITMAXFILELENGTH, AUDITFILE

AUDITFORMAT

Use this parameter to control the format of audit messages that are written to the console or audit file.

Parameter Syntax

```
AUDITFORMAT format
```

Arguments

format

a number representing a bit mask controlling the format options. Please see parameter LOGFORMAT for the bit mask.

Default

The default log format is 93 (date, time, milliseconds, process ID and log level)

Example

Display date, time, milliseconds only:

```
AUDITFORMAT 13
```

Display date, time only:

```
AUDITFORMAT 5
```

Considerations

- Audit messages will depend on the run mode – see parameter AUDITLEVEL for details

See also:

AUDITCONSOLE, AUDITFILE, AUDITLEVEL

AUDITLEVEL

Use this parameter to control what audit messages are written to the audit console or file.

Parameter Syntax

```
AUDITLEVEL detail
```

Arguments

detail

a number representing the detail level

Default

The default audit level is 50

Considerations

Considerations

- Audit messages are written only for the following run modes: PROXYS,PROXYC,PROXY,MQS,MQC,ODBCMXS, FTPS.
- The following table describes how to set AUDITLEVEL for the various run modes.

Audit Level	Run Modes PROXYS,PROXYC,PROXY,MQS,MQC,ODBCMXS	Run Mode FTPS
10	Startup of NSSL	Startup of NSSL
30		Logon of user
50	Network events (connect, disconnect)	FTP operations
60		Network events (connect, disconnect)

80	Data flowing through NSSL: byte count only	
90	Data flowing through NSSL: full byte dump (see parameter AUDITASCIIONLY for details)	

- For PROXYS,PROXYC,PROXY,MQS,MQC,ODBCMXS we recommend 50 for basic auditing and 99 for extended auditing including full traffic log.

  Note: If set to 99, all data flowing through the network is dumped to the audit log. This could include confidential data or passwords so make sure to properly secure the audit log files.

- For FTPS mode, we recommend 50 for normal auditing

See also:

AUDITCONSOLE, AUDITFILE, AUDITFORMAT

AUDITMAXFILELENGTH

Use this parameter to control the maximum size of the audit file.

Parameter Syntax

```
AUDITMAXFILELENGTH length
```

Arguments

length

a number representing the maximum audit file length in kilobytes.

Max. 40.000 or 40 MB

Min 100

Default

The default length is 20 000 KB.

Considerations

- After the current audit file reaches the maximum size a log rollover will occur. The current audit file will be renamed by replacing the last character with a "2". A new file with the AUDITFILE name will be created for subsequent audit output.

See also:

AUDITFILE, AUDITLEVEL

CACERTS

Use this parameter to specify a certificate chain validating the server or client certificate given by the SERVCERT or CLIENTCERT parameter.

Parameter Syntax

```
CACERTS file1 [, file2, ...]
```

Arguments

file1, file2, ...

the designated files are DER encoded X.509 CA certificates.

Default

If omitted, NSSL will search for a single "CACERT" file on the default subvolume.

Example

```
CACERTS $DATA1.NSSL.MYCA
```

Considerations

- The first file on the list must contain a certificate signing the given server certificate. Subsequent files must contain certificates that sign the previous certificate in the list.
- During SSL handshake, the certificate chain will be sent along with the client or server certificate to the SSL communication partner
- If a value of * is used for CACERTS, it will be assumed that the client or server certificate is self-signed.
- A CA certificate for testing purposes is delivered as CACERT file on the NSSL installation subvolume to enable quick start installation. This test CA certificate signs the test server certificate contained in SERVCERT or CLIENTCERT.

See also:

SERVCERT, CLIENTCERT

CIPHERSUITES

Use this parameter to specify which cipher suites are admissible for an NSSL secure server.

Parameter Syntax

```
CIPHERSUITES suite [, suite, ...]
```

Arguments

suite

specifies a cipher suite. Currently the following cipher suites are supported by NSSL:

- 0.1: RSA-key-exchange + NO ENCRYPTION and MD5 hash
- 0.2: RSA-key-exchange + NO ENCRYPTION and SHA1 hash
- 0.4: RSA-key-exchange + RC4-128-bit encryption and MD5 (RC4-MD5)
- 0.5: RSA-key-exchange + RC4-128-bit encryption and SHA (RC4-SHA)
- 0.10: RSA-key-exchange + 3-DES encryption and SHA (DES-CBC3-SHA)
- 0.47: RSA-key-exchange + 128-bit AES encryption and SHA (AES128-SHA)
- 0.55: RSA-key-exchange + 256-bit AES encryption and SHA (AES256-SHA)

Default

If omitted, NSSL will use 0.4, 0.10, 0.5 as accepted cipher suites.

Considerations

- During the SSL handshake the SSL client will send a list of cipher suites that it supports. If you allow multiple cipher suites, NSSL will select the first one from the list that matches a browser-supported cipher suite.

Example

```
CIPHERSUITES 0.5, 0.10
```

Note: The cipher suites 0.1 and 0.2 will NOT encrypt the traffic, they will only authenticate the partners and provide message integrity checking. Please only use if encryption is not required.

CLIENTAUTH

Use this parameter to enforce SSL client authentication when running as SSL server. The CLIENTAUTH parameter specifies a file (or a set of files) containing certificates. The certificate(s) will be sent to the client during connection setup. The client will reply with its own client certificate which needs to be signed by one of the certificates configured with the CLIENTAUTH parameter.

Parameter Syntax

```
CLIENTAUTH file1 [, file2, ...]
```

Arguments

file1, file2, ...

DER encoded X.509 CA certificate(s) which sign the certificate to be sent by the SSL client to NSSL. If the SSL client cannot send such a certificate, connections setup will fail.

'*'

No certificate request will be sent to the client

Default

If omitted, '*' is used and NSSL will not enforce SSL client authentication when running as SSL server.

Example

```
CLIENTAUTH $DATA1.NSSL.CACERT
```

Note: Prior to NSSL version 1043, that parameter was called TRUST. For downward compatibility, NSSL will copy the value of the TRUST parameter into the CLIENTAUTH parameter when running as SSL server.

CLIENTCERT

Use this parameter to specify the client certificate a secure NSSL server should use to authenticate itself to an SSL server.

Parameter Syntax

```
CLIENTCERT file
```

Arguments

file

Guardian file name of a DER encoded X.509 client certificate.

Default

If omitted or set to *, NSSL will not authenticate itself to the SSL server.

Example

```
CLIENTCERT $DATA1.NSSL.CLNTCERT
```

Considerations

- This parameter only applies to the run modes PROXYC and MQC, it will be ignored in other run modes
- A client certificate for testing purposes is delivered as CLNTCERT file on the NSSL installation subvolume to enable quick start installation.
- As NSSL indirectly supports BASE64 encoded certificates (after conversion to DER format using Crystal Point's certificate tools, see chapter "The Certificate Tools" in chapter "SSL Reference"), any client certificates received by a CA such as Verisign or Thawte can be used with with NSSL.
- The client certificate must match the private key file specified by CLIENTKEY.

See also

- "To have NSSL send a certificate to the SSL server" in chapter "SSL Reference".
- CLIENTKEY, CLIENTKEYPASS

CLIENTKEY

Use this parameter to specify the file containing the private key associated with the public key contained in the client certificate (CLIENTCERT).

Parameter Syntax

```
CLIENTKEY file
```

Arguments

file

file name of a DER encoded PKCS-8 encrypted private key file with PKCS-5 algorithm identifiers.

Default

If omitted, NSSL will search for a "CLIENTKEY" file on the default subvolume.

Example

```
CLIENTKEY $DATA1.NSSL.MYKEY
```

Considerations

- This parameter only applies to the run modes PROXYC and MQC, it will be ignored in other run modes
- The private key data in the file is password encrypted. For NSSL to be able to decrypt the file, the correct password must be specified by the CLIENTKEYPASS parameter.
- A private key file for testing purposes is delivered as "CLNTKEY" file on the NSSL installation subvolume to enable quick start installation. This private key file matches the test client certificate delivered as "CLNTCERT". The password for the CLNTKEY file is "test".

See also

CLIENTCERT, CLIENTKEYPASS

CLIENTKEYPASS

Use this parameter to specify the password for the file containing the private key associated with the public key given in the client certificate (see param CLIENTCERT).

Parameter Syntax

```
CLIENTKEYPASS password
```

Arguments

password

the password or pass phrase to decrypt the private key file. The password string may contain spaces. However, leading or trailing spaces will be ignored.

Default

If omitted, NSSL will try "test" as password.

Example

```
CLIENTKEYPASS my private passphrase
```

Considerations

- This parameter only applies to the run modes PROXYC and MQC, it will be ignored in other run modes
- The default password ("test") enables quickstart installation with the "CLIENTKEY" public key file delivered with NSSL.

See also

CLIENTCERT, CLIENTKEY

CONFIG

Use this parameter to specify a configuration file for an NSSL process.

Parameter Syntax

```
CONFIG file
```

Arguments

file

the name of the configuration file.

Default

If omitted, NSSL will not use a configuration file.

Example

```
CONFIG $DATA1.NSSL.SSLCONF
```

Considerations

- This parameter can only be specified as PARAM or on the startup line. It is not valid within a configuration file.
- Parameters specified in the configuration file can be overwritten by PARAM or startup line settings.

CONFIG2

Use this parameter to specify a second configuration file for an NSSL process.

Parameter Syntax

```
CONFIG2 file2
```

Arguments

file2

the name of the second configuration file.

Default

If omitted, NSSL will not use a second configuration file.

Example

```
CONFIG2 $DATA1.NSSL.SSLCONF2
```

Considerations

- Having a second configuration file for instance allows to store the pass phrases in a separate file with higher security settings
- The second configuration file has precedence over the first one
- This parameter can only be specified as PARAM or on the startup line. It is not valid within a configuration file.
- Parameters specified in the configuration file can be overwritten by PARAM or startup line settings.

CONTENTFILTER

Use this parameter to configure a text file with rules which will be applied to all incoming messages in run modes TELNETS, PROXYS, PROXYC and PROXY. If a message does not match the rule set, the connection will be terminated and the message will be discarded.

Parameter Syntax

```
CONTENTFILETER file
```

Arguments

file

The filename of the rule set file or * for no filtering.

Default

If omitted, NSSL will use a value of * (no filtereing).

Example

```
CONTENTFILTER CFILTER
```

Considerations

- The value of the parameter can be changed without stopping NSSL using the NSSLCOM command SET CONTENTFILTER file.
- The following example shows the syntax of the filter rules. This example will only allow messagest starting with "<A" or "<B" and ending with ">" to pass the filter.

```
################################################################################
##########
# file to define content filter rules
# empty lines or lines starting with '#' are ignored
################################################################################
##########

################################################################################
##########
# example file based on the following requirements:
#
# the following two are valid messages (double quotes are *not* part of
msg)
#   "<ABC>"
#   "<BBC>"
#
# the following two are *not* valid messages
#   "<CCC>"    - does not start with "<A" or "<B"
#   "text"     - does not start with "<"
################################################################################
##########

################################################################################
##########
# msg delimiters (required)
# used to define a "message" as part of the byte stream
# all bytes are ASCII values represented as decimal numbers
################################################################################
##########
# start with < sign = 3C hex = 60 dec
msgstartbyte 60
# end with > sign = 3E hex = 62 dec
msgendbyte 62

################################################################################
##########
# list of regular expressions, in double quotes
# (at least one required)
#
# note that the engine implements "traditional unix regular
expressions"
# see
#
en.wikipedia.org/wiki/Regular_expression#Traditional_Unix_regular_expre
ssions
# for details
#
# regular expressions are combined using an implicit "logical or"
# a message matching any single regular expression will pass
# a message matching no regular expression will fail
# at least one regular expression must be present
################################################################################
##########
# allow any message starting with "<A"
regexp "^<A."
# allow any message starting with "<B"
regexp "^<B."
```

DENYIP

Use this parameter to specify which remote IP addresses are to be forbidden to establish sessions ("black list").

Parameter Syntax

```
DENYIP range
```

Arguments

range

specifies a set of forbidden remote IP addresses. Valid values are

- single IP addresses such as 10.1.1.11
- a subnet such as 10.1.1.*
- comma-separated lists of single IP addresses and subnets

Default

If omitted, NSSL will use an empty entry to not forbid any remote IP addresses

Example

```
DENYIP 10.0.5.*, 10.0.6.123
```

Considerations

- See section "Using NSSL to limit the remote IP addresses" (in chapter "Introduction") for the concept of remote IP filtering
- The paramter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.

DONOTWARNONERROR

Use this parameter to log selected errors with LOGLEVEL 20 rather than as WARNING. By default, all errors on sockets result in a WARNING being displayed in the NSSL log. Using this parameter, a log message with LOGLEVEL 20 will be issued instead for the configured error numbers.

Parameter Syntax

```
DONOTWARNONERROR ErrorList
```

Arguments

ErrorList

specifies a list of comma-separated error numbers

Default

If omitted, NSSL will use an empty entry.

Example

```
DONOTWARNONERROR 4120
```

Considerations

- The example shown will yield in error 4120 ("Connection reset by remote") generating a log message with LOGLEVEL 20 rather than a WARNING.

FTPALLOWPLAIN

Use this parameter to specify whether NSSL will allow unencrypted FTP sessions when running in FTPS mode.

Parameter Syntax

```
FTPALLOWPLAIN boolean
```

Arguments

boolean

if set to TRUE or 1 or Yes, NSSL will allow unencrypted traffic

Default

If omitted, NSSL will *not* allow unencrypted traffic

Example

```
FTPALLOWPLAIN TRUE
```

Considerations

- This parameter is relevant only if NSSL is running in the FTPS mode.

FTPCALLOW200REPLY

Use this parameter to specify whether NSSL will allow an illegal "200" response to the AUTH TLS command sent to the remote FTP/TLS server.

Parameter Syntax

```
FTPCALLOW200REPLY boolean
```

Arguments

boolean

if set to TRUE or 1 or Yes, NSSL will allow the illegal response.

Default

If omitted, NSSL will *not* allow the illegal 200 response.

Example

```
FTPCALLOW200REPLY TRUE
```

Considerations

- This parameter is relevant only if NSSL is running in the FTPC mode.
- The FTP/TLS specfication requires a "234" reply code to the AUTH TLS command. To support some older FTP/TLS implementations to run against NSSL in FTPC mode, this parameter has been added

FTPLOCALDATAPORT

Use this parameter to specify how NSSL will pick the local data port for the data connection in FTPC mode with PASSIVE set to true.

Parameter Syntax

```
FTPLOCALDATAPORT number
```

Arguments

number

0 for “pick a random port” or any specific port number

Default

If omitted, a value of 0 will be used.

Example

```
FTPLOCALDATAPORT 20
```

Considerations

- This parameter is relevant only if NSSL is running in the FTPC mode with PASSIVE set to TRUE
- Choosing a value other than zero will be firewall-friendly, however it can result in errors if the remote FTP server does not choose random data ports itself.

FTPMAXPORT

Use this parameter to specify the maximum port number NSSL will use for FTP data connections

Parameter Syntax

```
FTPMAXPORT number
```

Arguments

number

the maximum port number NSSL will use for FTP data connections

Default

If omitted, NSSL will use a value of 41000

Example

```
FTPMAXPORT 22000
```

Considerations

- This parameter is relevant only if NSSL is running in the FTPS or FTPC mode.
- Together with the parameter FPTMinPORT it controls the values NSSL assigns for the FTP data sockets. You can change this value to make sure that the FTP data connections will not interfere with other TCP/IP services on your system.

FTPMINPORT

Use this parameter to specify the minimum port number NSSL will use for FTP data connections

Parameter Syntax

```
FTPMINPORT number
```

Arguments

number

the minimum port number NSSL will use for FTP data connections

Default

If omitted, NSSL will use a value of 40000

Example

```
FTPMINPORT 20000
```

Considerations

- This parameter is relevant only if NSSL is running in the FTPS or FTPC mode.
- Together with the parameter FPTMAXPORT it controls the values NSSL assigns for the FTP data sockets. You can change this value to make sure that the FTP data connections will not interfere with other TCP/IP services on your system.

HTTPBASE

Use this parameter to define the subvolume an NSSL HTTP server should return requested resources from.

Parameter Syntax

```
HTTPBASE subvol | *
```

Arguments

subvol

the name of a GUARDIAN subvolume.

*

means that the subvolume NSSL has been started on is used.

Default

If omitted, the standard installation subvolume for NSSL HTML contents will be used ("NSSLHTML")

Considerations

- The parameter is ignored if not operating as an HTTP server.

Important Note: Crystal Point strongly recommends using a separate subvolume other than the NSSL installation subvolume as HTTPBASE. Do not put any files on this subvolume other than those allowed to be downloaded by a browser user. Having any private key files on this subvolume will compromise security!

Example

```
HTTPBASE $DATA1.MYHTML
```

HTTPZIP

Use this parameter to specify a ZIP file an NSSL HTTP server should return requested resources from.

Parameter Syntax

```
HTTPZIP file
```

Arguments

file

the name of a ZIP file containing HTTP contents.

Default

If omitted, an NSSL http server will search for a file named "HTTPZIP" on its default subvolume. NSSL is delivered with a HTTPZIP archive containing the NSSL welcome page and this manual in HTML format for on-line reading.

Considerations

- This option allows easy deployment of web content with a NonStop Guardian system despite of the limitations of the Guardian file system.
- To prepare web content to be deployed with NSSL just use a standard ZIP tool to pack all required files into a ZIP archive. As NSSL does not support compression, please make sure to switch it off when packing your archive. The archive must be uploaded to the NonStop server in binary format.
- The parameter is ignored if not operating as an HTTP server.

INTERFACE

Use this parameter to specify the IP address NSSL should use for local binding on incoming connections.

Parameter Syntax

```
INTERFACE ip address
```

Arguments

ip address

the IP address to bind to or “*” for none

Default

If omitted, NSSL will use the value of “*” and bind to no specific IP address

Example

```
INTERFACE 10.0.0.197
```

Considerations

- The parameter is relevant for the following run modes: HTTP, HTTPS, PROXY (incoming socket), PROXYS (incoming socket), PROXYC (incoming socket), FTPS (control listening socket being connected to from remote FTP client), FTPC (control listening socket being connected to from local NonStop FTP client)
- Use this parameter to control which IP address NSSL binds to for incoming connections.
- If a host name rather than an IP address is used to configure INTERFACE, name resolution will take place only once during startup. If name resolution fails, NSSL will terminate during startup
- See parameter TARGETINTERFACE for additional information.

KEEPALIVE

Use this parameter to specify if keep alive messages should be sent to the TCP/IP sockets of established links.

Parameter Syntax

```
KEEPALIVE mode
```

Arguments

mode

- 1 (on) for sending keep alive messages
- 0 (off) no messages are sent

Default

By default, keep alive messages are sent (1).

LICENSE

Use this parameter to specify different location for the NSSL license file.

Parameter Syntax

```
LICENSE file
```

Arguments

file

the filename of the NSSL license file

Considerations

- If the filename is not fully qualified, NSSL will add the home subvolume of the object file to the file name.

Default

If omitted, an NSSL process will search for a file named "LICENSE" on its default subvolume (i.e. where the NSSL object resides).

LOGCONSOLE

Use this parameter to define if and to what console device NSSL log messages are written to.

Parameter Syntax

```
LOGCONSOLE * | % | $0 | logdevice
```

Arguments

*

means that no log messages are written to a console

%

means that log messages are written to the home terminal of the NSSL process

$0

log messages are written to $0

logdevice

log messages are written the given device (e.g. $DEV.#SUBDEV)

Considerations

- The LOGLEVEL parameter controls what messages are produced by NSSL.
- The paramter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.

Default

By default, log messages will be written to the home terminal ("%")

See also:

LOGEMS, LOGFILE, LOGLEVEL

LOGEMS

Use this parameter to define if NSSL log messages are written to EMS.

Parameter Syntax

```
LOGEMS collector | *
```

Arguments

`*`

means that no log messages are written to EMS

`collector`

Means that log messages are written to the collector with that name

Default

By default, no log messages are written to EMS ("*")

Considerations

- The LOGLEVEEMS parameter controls what messages are produced by NSSL.
- The LOGFORMATEMS parameter controls the log message format.
- The parameter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.
- To send messages to the default collector $0 use
  LOGEMS $0
- If the EMS collector can not be opened during startup, NSSL will terminate. If the EMS collector can not be opened after changing it through NSSLCOM, the old collector will stay active

See also:

LOGLEVELEMS, LOGFORMATEMS, LOGMAXFILELENGTH, LOGFILERETENTION

LOGFILE

Use this parameter to define if and to what file NSSL log messages are written.

Parameter Syntax

```
LOGFILE * | file
```

Arguments

*

means that no log messages are written to a file

file

the name of the log file

Default

By default, no log messages are written to a file ("*")

Considerations

- The LOGLEVEL parameter controls what messages are produced by NSSL.
- The LOGFORMAT parameter controls the log message format.
- The parameter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.
- See section “Logfile/Auditfile rollover using round robin” in chapter "Configuring And Running NSSL" for details on logfile rollover.

See also:

LOGLEVELFILE, LOGFORMATFILE, LOGMAXFILELENGTH, LOGFILERETENTION

LOGFILERETENTION

Use this parameter to control how many log files NSSL keeps when logfile rollover occurs

Parameter Syntax

```
LOGFILERETENTION n
```

Arguments

n

number of log files to keep

Default

By default, 10 files are kept.

Considerations

- a minimum of 10 is enforced for that parameter
- See section “Logfile/Auditfile rollover using round robin” in chapter "Configuring And Running NSSL" for details on logfile rollover.

See also:

LOGMAXFILELENGTH, LOGFILE

LOGFORMAT

Considerations

- This parameter is retained for downward-compatibility only and has been replaced by the parameters LOGFORMATCONSOLE and LOGFORMATFILE.
- If no value is set for the parameters LOGFORMATCONSOLE or LOGFORMATFILE, they will inherit their value from the parameter LOGFORMAT.
- If both LOGFORMATCONSOLE and LOGFORMATFILE are set with a value, the parameter of LOGFORMAT becomes meaningless.

See also:

LOGFORMATCONSOLE, LOGFORMATEMS, LOGFORMATFILE

LOGFORMATCONSOLE

Use this parameter to control the format of the log messages that are written to the console.

Parameter Syntax

```
LOGFORMAT format
```

Arguments

format

a number representing a bit mask controlling the following format options:

bit 1 (decimal 1):	Date
bit 2 (decimal 2):	header (log messages a pre-fixed with "[log]")
bit 3 (decimal 4):	Time
bit 4 (decimal 8):	Milliseconds
bit 5 (decimal 16):	Process ID (name or PIN)
Bit 7 (decimal 64)	Log Level of Message

Default

The default log format is 93 (date, time, milliseconds, process ID and log level)

Example

Display date, time, milliseconds only:

```
LOGFORMAT 13
```

Display date, time only:

```
LOGFORMAT 5
```

See also:

LOGFORMATEMS, LOGFORMATFILE

LOGFORMATEMS

Use this parameter to control the format of the log messages that are written to EMS.

Parameter Syntax

```
LOGFORMATEMS format
```

Arguments

format

a number representing a bit mask controlling the following format options:

bit 1 (decimal 1)	Date
bit 2 (decimal 2)	header (log messages a pre-fixed with "[log]")
bit 3 (decimal 4)	Time
bit 4 (decimal 8)	Milliseconds
bit 5 (decimal 16)	Process ID (name or PIN)
bit 7 (decimal 64)	Log Level of Message

Default

The default log format is 93 (date, time, milliseconds, process ID and log level).

Example

Display date, time, and milliseconds only:

```
LOGFORMATEMS 13
```

Display date, time only:

```
LOGFORMATEMS 5
```

See also:

LOGFORMATCONSOLE, LOGFORMATFILE

LOGFORMATFILE

Use this parameter to control the format of the log messages that are written to the log file.

Parameter Syntax

```
LOGFORMATFILE format
```

Arguments

format

a number representing a bit mask controlling the following format options:

bit 1 (decimal 1)	Date
bit 2 (decimal 2)	header (log messages a pre-fixed with "[log]")
bit 3 (decimal 4)	Time
bit 4 (decimal 8)	Milliseconds
bit 5 (decimal 16)	Process ID (name or PIN)
bit 7 (decimal 64)	Log Level of Message

Default

The default log format is 93 (date, time, milliseconds, process ID and log level).

Example

Display date, time, milliseconds only:

```
LOGFORMAT 13
```

Display date, time only:

```
LOGFORMAT 5
```

See also:

LOGFORMATCONSOLE

LOGLEVEL

Considerations

- This parameter is retained for downward-compatibility only and has been replaced by the parameters LOGLEVELCONSOLE and LOGLEVELFILE.
- If no value is set for the parameters LOGLEVELCONSOLE or LOGLEVELFILE, they will inherit their value from the parameter LOGLEVEL.
- If both LOGLEVELCONSOLE and LOGLEVELFILE are set with a value, the parameter of LOGLEVEL becomes meaningless.

See also:

LOGLEVELCONSOLE, LOGLEVELEMS, LOGLEVELFILE

LOGLEVELCONSOLE

Use this parameter to control what messages are written to the log console.

Parameter Syntax

```
LOGLEVELCONSOLE detail
```

Arguments

detail

a number representing the detail level

Default

For downward compatibility, the default log level is taken from the parameter LOGLEVEL if present. If no parameter LOGLEVEL is present, a default of 50 is used.

Considerations

- Using the parameter LOGLEVELCONSOLE allows to set a different log level for the output to LOGCONSOLE than for the output to LOGFILE.
- The parameter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.

See also:

LOGCONSOLE, LOGLEVELFILE, LOGFORMATCONSOLE, LOGLEVELEMS

LOGLEVELEMS

Use this parameter to control what messages are written to EMS.

Parameter Syntax

```
LOGLEVELEMS detail
```

Arguments

detail

a number representing the detail level

Default

For downward compatibility, the default log level is taken from the parameter LOGLEVEL if present. If no parameter LOGLEVEL is present, a default of 50 is used.

Considerations

- Different log levels can be used for the outputs to LOGCONSOLE, LOGLEVELEMS, and LOGFILE.
- The parameter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.

See also:

LOGEMS, LOGLEVELCONSOLE, LOGLEVELFILE, LOGMAXFILELENGTH, LOGFORMATFILE

LOGLEVELFILE

Use this parameter to control what messages are written to the log file.

Parameter Syntax

```
LOGLEVELFILE detail
```

Arguments

detail

a number representing the detail level

Default

For downward compatibility, the default log level is taken from the parameter LOGLEVEL if present. If no parameter LOGLEVEL is present, a default of 50 is used.

Considerations

- Using the parameter LOGLEVELFILE allows to set a different log level for the output to LOGFILE than for the output to LOGCONSOLE.
- The parameter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.

See also:

LOGFILE, LOGLEVELCONSOLE, LOGLEVELEMS , LOGMAXFILELENGTH, LOGFORMATFILE

LOGMAXFILELENGTH

Use this parameter to control the maximum size of a log file.

Parameter Syntax

```
LOGMAXFILELENGTH length
```

Arguments

length

a number representing the maximum log file length in kilobytes.

Max. 40.000 or 40 MB

Min 100

Default

The default length is 20 000 KB.

Considerations

- After the current file reaches the maximum size a log rollover will occur. The current log file will be renamed by replacing the last character with a "2". A new file with the LOGFILE name will be created for subsequent log output.

See also:

LOGFILE, LOGLEVEL

LOGMEMORY

Use this parameter to send the memory usage of NSSL to the log output in regular intervals

Parameter Syntax

```
LOGMEMORY number_of_io's
```

Arguments

number_of_io's

a number representing after how many I/O operations NSSL will send its memory usage to the log output

Default

The default is 0 meaning that memory usage will not be logged

Considerations

- Use to have an easy correlation between memory usage of NSSL and events in the log output. Do not use if memory usage of NSSL is not of interest for you.
- The paramter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section “Command Interface NSSLCOM” for details.

MAXSESSIONS

Use this parameter to limit the number of concurrent connections in run modes TELNETS, PROXYS, PROXYC and PROXY.

Parameter Syntax

```
MAXSESSIONS max
```

Arguments

max

the number of allowed concurrent sessions or 0 for unlimited.

Default

If omitted, NSSL will.use a value of 0 (no limits)

Example

```
MAXSESSIONS 100
```

Considerations

- If the number of allowed sessions is reached, any further connection request will be rejected and a Warning will be written to the log file
- The current number of connections is displayed in the STATUS command of NSSLCOM.

MAXVERSION

Use this parameter to define the maximum admissible SSL/TLS protocol version.

Parameter Syntax

```
MAXVERSION version
```

version

an SSL/TLS version number. Currently supported values are:

- 2.0: SSL 2.0
- 3.0: SSL 3.0
- 3.1: SSL 3.1 / TLS 1.0

Default

The default for this parameter is "3.1" (i.e. SSL 3.1 / TLS 1.0).

See also:

MINVERSION

MINVERSION

Use this parameter to define the minimum admissible SSL/TLS protocol version.

Parameter Syntax

```
MINVERSION version
```

version

an SSL/TLS version number. Currently supported values are:

- 2.0: SSL 2.0
- 3.0: SSL 3.0
- 3.1: SSL 3.1 / TLS 1.0

Default

The default for this parameter is "3.1"

Considerations

- For security reasons, it is recommended to use the latest version of the TLS protocol as standardized by the IETF (3.1). This requires setting MINVERSION to "3.1".

See also:

MAXVERSION

PASSIVE

Use this parameter to define the direction of the data socket connection in FTPC mode

Parameter Syntax

```
PASSIVE mode
```

Arguments

mode

- 1 for passive mode
- 0 for active mode

Default

The default for this parameter is 1 for passive mode

Considerations

- This parameter is only relevant in the FTPC run mode of NSSL
- In FTP, the data socket connection request can be made by the FTP client ("passive mode") or by the FTP server ("active mode"). The best choice for your environment depends on the capabilities of the FTP server you are communicating with and on your firewall settings
- NSSL in FTPS mode currently only supports passive mode, therefore to interact with NSSL in FTPS mode, make sure to set the PASSIVE parameter to 1 for NSSL running in FTPC mode.

PEERCERTCOMMONNAME

Use this parameter to enforce verification of the content of remote certificates presented to NSSL.

Parameter Syntax

```
PEERCERTCOMMONNAME commonname
```

Arguments

commonname

the expected common name of the remote certificate

Default

The default for this parameter is '*' which means the content will not be verified.

Examples

```
PEERCERTCOMMONNAME tandem1.mycompany.com
```

Considerations

- This parameter should not be used together with the parameter PEERCERTFINGERPRINT as behavior may be unpredictable then.
- If other than '*', the actual common name of the remote certificate will be compared against the content of the parameter.
- If the actual value of the common name in the remote certificate is *part of* the value configured in the parameter, it will be accepted. This allows to configure a list of common names.
- If the matching fails, no sessions can be established

PEERCERTFINGERPRINT

Use this parameter to enforce verification of the content of remote certificates presented to NSSL.

Parameter Syntax

```
PEERCERTFINGERPRINT fingerprint
```

Arguments

fingerprint

the expected fingerprint of the remote certificate

Default

The default for this parameter is '*' which means the content will not be verified.

Examples

```
PEERCERTFINGERPRINT b533d676f9538617484bd4302c8db70e
```

Considerations

- This parameter should not be used together with the parameter PEERCERTCOMMONNAME as behavior may be unpredictable then.
- If other than '*', the actual content of the remote certificate will be compared against the content of the parameter.
- If the actual value in the certificate is part of the value configured in the parameter, it will be accepted. This allows to configure a list of fingerprints or common names.

- Fingerprints will be compared both as MD5 and SHA1 hashes.
- If the matching fails, no sessions can be established.

PORT

Use this parameter to specify the port number an NSSL server should listen for incoming connections.

Parameter Syntax

```
PORT number
```

Arguments

number

the decimal number of a TCP/IP port.

Default

The default for this parameter depends on the NSSL run mode:

HTTP:	80
HTTPS:	443
TELNETS:	11011 (*)
PROXYS	11011 (*)
PROXYC	11012 (*)
FTPS	11013 (*)
FTPC	11014 (*)
FTPCPLAIN	11014 (*)
ATTUNITYS	11015 (*)
MQS	1414
MQC	11414

Considerations

- If operating as a secure server, NSSL will only accept SSL connections on the specified port.
- Starting NSSL to listen on a port number <=1024 requires SUPER group access.
- The ICANN manages a list of "well-known" port numbers for various protocols (see http://www.iana.org/assignments/port-numbers). Most run modes of NSSL can not be mapped against this list with certainty, those run modes are marked with an asterisk (*). The default ports for those run modes were chosen from an "unassigned" port range (11002-11110)
- The choice for the PORT value in your environment will depend on the applications already running on your NonStop systems and the ports they use as well as your firewall configuration.
- You can specify a comma-separated list of multiple ports, see section "Multiple Configurations in a Single NSSL Process" for details.

PTCPIPFILTERKEY

Use this parameter to specify a filter key to enable round robin filtering with Parallel Library TCP/IP or TCP/IPV6.

Parameter Syntax

```
PTCPIPFILTERKEY password | *
```

Arguments

password

a password serving as a key to enable round robin filtering for multiple instances of NSSL servers listening on the same port. The password will override the value of the DEFINE =PTCPIP^FILTER^KEY, which may have been passed to NSSL at startup.

*

No filter key will be set. However, any DEFINE =PTCPIP^FILTER^KEY passed to NSSL at startup will remain in effect.

Default

The default for this parameter is *.

Considerations

- Use this parameter to enable round robin filtering for multiple NSSL servers configured as generic processes (DEFINEs cannot be propagated to generic processes).

SERVCERT

Use this parameter to specify the server certificate a secure NSSL server should use to authenticate itself to an SSL client.

Parameter Syntax

```
SERVCERT file
```

Arguments

file

Guardian file name of a DER encoded X.509 server certificate.

Default

If omitted, NSSL will search for a file "SERVCERT" on the default subvolume.

Example

```
SERVCERT $DATA1.NSSL.MYCERT
```

Considerations

- A server certificate for testing purposes is delivered as SERVCERT file on the NSSL installation subvolume to enable quick start installation.
- As NSSL indirectly supports BASE64 encodes certificates, any client certificates received by a CA such as Verisign or Thawte can be used with with NSSL.
- The server certificate must match the private key file specified by SERVKEY.

See also:

SERVCERT, SERVKEY

SERVKEY

Use this parameter to specify the private key file for a secure NSSL server.

Parameter Syntax

```
SERVKEY file
```

Arguments

file

file name of a DER encoded PKCS-8 encrypted private key file with PKCS-5 algorithm identifiers.

Default

If omitted, NSSL will search for a "SERVKEY" file on the default subvolume.

Example

```
SERVKEY $DATA1.NSSL.MYKEY
```

Considerations

- The private key data in the file is password encrypted. For NSSL to be able to decrypt the file, the correct password must be specified by the SERVKEYPASS parameter.
- A private key file for testing purposes is delivered as "SERVKEY" file on the NSSL installation subvolume to enable quick start installation. This private key file matches the test server certificate delivered as "SERVCERT". The password for the SERVKEY file is "test".

See also:

SERVCERT, SERVKEYPASS

SERVKEYPASS

Use this parameter to specify the password for the private key file.

Parameter Syntax

```
SERVKEYPASS password
```

Arguments

password

the password or pass phrase to decrypt the private key file. The password string may contain spaces. However, leading or trailing spaces will be ignored.

Default

If omitted, NSSL will try "test" as password.

Example

```
SERVKEYPASS my private passphrase
```

Considerations

- The default password ("test") enables quickstart installation with the "SERVKEY" public key file delivered with NSSL.

See also:

SERVCERT, SERVKEYPASS

SLOWDOWN

Use this parameter to make NSSL use less CPU cycles for encryption. This will result in a decrease of possible throughput.

Parameter Syntax

```
SLOWDOWN <ticks>
```

Arguments

ticks

After each I/O operation, NSSL will call the Guardian System Procedure DELAY with the value of <ticks>. A higher value will decrease both throughput and CPU usage of NSSL.

Default

If omitted, SLOWDOWN will be 0 and NSSL will consume all available CPU resources.

Example

```
SLOWDOWN 1
```

Considerations

- In most installations, the default value of 0 should be acceptable
- The parameter is mostly intended for use with the FTPC or FTPS modes of NSSL. Setting SLOWDOWN to values between 1 and 5 will significantly reduce CPU usage but will also make the time a file transfer will take higher.
- The impact of NSSL high volume data encryption/decryption can also be influenced by the priority of the NSSL process. However, if it is desirable to run NSSL at a higher priority than the target plain servers/clients, the SLOWDOWN can be used to limit the impact of the cryptographic operations.
- The best value for your environment will depend both on your hardware and requirements.

SOCKSHOST, SOCKSPORT, SOCKSUSER

Use these three parameter to make NSSL act as a SOCKS Version 4 client in the run modes FTPC, FTPCPLAIN, PROXY or PROXYC.
(The SOCKS protocol is a protocol that relays TCP sessions at a firewall host to allow application users transparent access across the firewall. For more information about SOCKS, please see http://en.wikipedia.org/wiki/SOCKS.)

Parameter Syntax

```
SOCKSHOST sockshost
SOCKSPORT socksport
SOCKSUSER socksuser
```

Arguments

sockshost

the hostname or IP address of the host running the SOCKS-Version 4 enabled firewall. A value of * indicates that the SOCKS protocol will not be used.

socksport

the listening port of the host running the SOCKS-Version 4 enabled firewall

socksuser

the SOCKS user name to be used to authenticate against the SOCKS server

Default

If omitted, NSSL will use a value of * for SOCKSHOST meaning the SOCKS protocol will not be used.

Example

```
SOCKSHOST 172.3.5.99
SOCKSPORT 1911
SOCKSUSER sockstest
```

Considerations

- In run modes PROXY and PROXYC the value of TARGETPORT will still be required to determine the final host to connect to.
- In run modes FTPC and FTPCPLAIN the final host to connect to will be configured by adding it to the user name just as when not using SOCKS.

SUBNET

Use this parameter to specify the TCP/IP process an NSSL process should listen on for incoming connections.

Parameter Syntax

SUBNET *tcpip-process-name*

Arguments

tcpip-process-name

the name of an existing TCP/IP process on your system

Default

If omitted, the NSSL process will be bound to "$ZTC0".

Example

```
SUBNET $ZTC03
```

Considerations

- If you added a DEFINE =TCPIP^PROCESS^NAME to the TACL environment you use to start NSSL, this setting will override the SUBNET parameter.
- If you use Parallel Library TCPIP and want to share identical ports across multiple instances of NSSL you need to add an identical DEFINE to all instances sharing that port as in the following example (please refer to the HP NonStop manual "TCP/IP (Parallel Library) Configuration and Management Manual—522271-002", section 3, subsection "Monolithic Listening Model" for more details):

```
ADD DEFINE =PTCPIP^FILTER^KEY, class map, file A1234
```

SWAPCOMSECURITY

Use this parameter to restrict execution of NSSLCOM commands

Parameter Syntax

```
SWAPCOMSECURITY boolean
```

Arguments

boolean

if set to TRUE, "sensitive" NSSLCOM commands can only be executed by
a) a member of the SUPER group
b) the user under which the NSSL process in running

Default

The default for this parameter is FALSE.

Example

```
SWAPCOMSECURITY TRUE
```

Considerations

- The following commands are considered sensitive:
  - all SET commands
  - the LOGMESSAGE, ROLLOVER LOGFILE and RELOAD CERTIFICATES commands

TARGETINTERFACE

Use this parameter to specify the IP address NSSL should use for local binding on outgoing connections.

Parameter Syntax

```
TARGETINTERFACE ip address
```

Arguments

ip address

the IP address to bind to or “*” for none

Default

If omitted, NSSL will use the value of “*” and bind to no specific IP address

Example

```
TARGETINTERFACE 10.0.0.197
```

Considerations

- The parameter is relevant for the following run modes: PROXY (outgoing socket), PROXYS (outgoing socket), PROXYC (outgoing socket), FTPS (control socket connecting to FTPSERV), FTPC (control socket connecting to remote FTP server)
- Use this parameter to control which IP address NSSL binds to for outgoing connections.

- If a host name rather than an IP address is used to configure TARGETINTERFACE, name resolution will take place only once during startup. If name resolution fails, NSSL will terminate during startup
- See parameter INTERFACE for additional information.

TARGETHOST

Use this parameter to specify the IP host an NSSL proxy server should route connections to.

Parameter Syntax

```
TARGETHOST ip address
```

Arguments

ip address

the IP address of the target host

Default

If omitted, the NSSL proxy route connections to the "local loopback address" ("127.0.0.1").

Example

```
TARGETHOST 192.45.23.3
```

Considerations

- The parameter must not be given if NSSL operates as a HTTP(S) server, in this case NSSL abends with an error message .
- If the target server process runs on the same TCP/IP process (SUBNET) you should use the "local loopback address" ("127.0.0.1"). This is recommended for proxy servers, as it avoids that unencrypted data has to traverse the network.
- You can specify a comma-separated list of multiple target hosts, see section "Multiple Configurations in a Single NSSL Process" for details

TARGETPORT

Use this parameter to specify the port number an NSSL proxy server should route connections to.

Parameter Syntax

```
TARGETPORT number
```

Arguments

number

the decimal number of the target TCP/IP port.

Default

If omitted, the NSSL proxy will try route connections to the well known telnet port (23).

Example

```
TARGETPORT 1023
```

Considerations

- The parameter is ignored in the run modes HTTP, HTTPS and FTPC.
- You can specify a comma-separated list of multiple target ports, see section "Multiple Configurations in a Single NSSL Process" for details.

TARGETSUBNET

Use this parameter to specify the TCP/IP process an NSSL process should use for outgoing connections.

Parameter Syntax

```
TARGETSUBNET tcpip-process-name
```

Arguments

tcpip-process-name

the name of an existing TCP/IP process on your system

Default

If omitted, the NSSL process will use same TCP/IP process which is configured for incoming connections (SUBNET parameter).

Example

```
TARGETSUBNET $ZTC03
```

Considerations

- If you added a DEFINE =TCPIP^PROCESS^NAME to the TACL environment you use to start NSSL, this setting will override the TARGETSUBNET parameter.
- The parameter is ignored in the run modes HTTP, HTTPS

TCPIPHOSTFILE

Use this parameter to specify the value of the DEFINE=TCPIP^HOST^FILE value.

Parameter Syntax

```
TCPIPHOSTFILE hostfile | *
```

Arguments

hostfile

a hostfile to be used for DNS name resolution. The hostfile will override the value of the DEFINE =TCPIP^HOST^FILE, which may have been passed to NSSL at startup.

*

No hostfile will be set. However, any DEFINE =TCPIP^HOST^FILE passed to NSSL at startup will remain in effect.

Default

The default for this parameter is *.

Considerations

- See the HP NonStop manual for details of the usage of the DEFINE =TCPIP^HOST^FILE.

TCPIPNODEFILE

Use this parameter to specify the value of the DEFINE=TCPIP^NODE^FILE value.

Parameter Syntax

```
TCPIPNODEFILE nodefile | *
```

Arguments

nodefile

a nodefile to be used for DNS name resolution. The nodefile will override the value of the DEFINE =TCPIP^NODE^FIL., which may have been passed to NSSL at startup.

*

No nodefile will be set. However, any DEFINE =TCPIP^NODE^FIL. passed to NSSL at startup will remain in effect.

Default

The default for this parameter is *.

Considerations

- See the HP NonStop manual for details of the usage of the DEFINE =TCPIP^NODE^FILE.

TCPIPRESOLVERNAME

Use this parameter to specify the value of the DEFINE =TCPIP^RESOLVER^NAME value.

Parameter Syntax

```
TCPIPRESOLVERNAME resolver | *
```

Arguments

resolver

a resolver to be used for DNS name resolution. The resolver will override the value of the DEFINE =TCPIP^RESOLVER^NAME, which may have been passed to NSSL at startup.

*

No resolver will be set. However, any DEFINE =TCPIP^RESOLVER^NAME passed to NSSL at startup will remain in effect.

Default

The default for this parameter is *.

Considerations

- See the HP NonStop manual for details of the usage of the DEFINE =TCPIP^RESOLVER^NAME.

TCPNODELAY

Use this parameter to specify whether RFC1323 will be activated on all sockets which NSSL controls.

Parameter Syntax

```
TCPNODELAY boolean
```

Arguments

boolean

if set to TRUE or 1 or Yes, NSSL will activate RFC1323.

Default

If omitted, NSSL will *not* activate RFC1323.

Example

```
TCPNODELAY TRUE
```

Considerations

- If this parameter is set to true, NSSL sets a socket option TCP_NODELAY when initializing sockets. This can help speed up throughput – please see RFC1323 and the HP NonStop TCP/IP programming manual for details.

TRUST

Use this parameter to specify a list of trusted CAs when running as SSL client.

Parameter Syntax

```
TRUST fingerprint [, fingerprint, ...]
```

or

```
TRUST certificate [, certificate, ...]
```

Arguments

fingerprint

the trusted CA certificate's MD5 fingerprint.

certificate

the trusted CA certificate in DER encoded format

Default

If omitted, NSSL will not check the TLS/SSL partner's certificate chain.

Examples

```
TRUST b533d676f9538617484bd4302c8db70e, a723502b68675b667ebde9964ae29543
TRUST rootcert
```

Considerations

- The TRUST parameter can be specified in two ways: either by specifying the MD5 fingerprints of the CA certificates or by specifying a filename containing the full certificate in DER encoding. The two formats can not be mixed.

- NSSL versions 1043 and earlier only support specifying the fingerprints of the trusted certificates
- If the remote SSL server is sending the complete certificate chain, the two forms of specifying the trusted CAs do not differ in functionality. Some SSL servers do not send the complete certificate chain during the handshake; for those servers the missing signing certificate(s) should be specified with the "certificate" syntax of the parameter
- The paramter can be changed without having to restart NSSL using the NSSLCOM command interpreter, please see section "Command Interface NSSLCOM" for details

Note: Prior to NSSL version 1043, that paramter was also used to enforce SSL client authentication when running as SSL server. This is now done with a new parameter CLIENTAUTH.

Multiple Configurations in a Single NSSL Process

A single NSSL process can listen on multiple ports at once and forward them to different IP addresses/port numbers. The following parameters are global to a single NSSL instance:

- SUBNET
- TARGETSUBNET
- run mode

The following three parameters can be supplied as comma-separated lists:

- PORT
- TARGETPORT
- TARGETHOST

In case a comma-separeted list is found, NSSL will match the individual entries to create tupels (PORT, TARGETPORT, TARGETHOST). Incoming connections on each PORT will then be forwarded to the matching TARGETPORT and TARGETHOST.

As an example, if you want to forward

- connections coming in on port 1023 to port 1023 on host Host23
- connections coming in on port 1024 to port 1024 on host Host24

you would start NSSL as follows:

```
RUN NSSL PROXYS; PORT 1023,1024; TARGETPORT 23,24; TARGETHOST Host23,Host24
```

Non-Stop Availability

Using NSSL ensures non-stop availability of NonStop based applications across the network in conjunction with a OutsideViewWEB or AppView deployment. Running on the Guardian platform, NSSL takes advantage of the NonStop fundamentals.

On G series systems, NSSL services can be configured as generic processes, enabling automatic recovery from failures, such as CPU outages. For D series systems, non-stop availability can be achieved by implementing NSSL services as static PATHWAY servers monitored by a non-stop Pathway Monitor.

Note: NSSL cannot be run as a non-stop process. However, this is not required to achieve non-stop availability. Running as a non-stop process would not add value, as TCP sessions (both for TELNET and HTTP) are reset upon CPU takeover. non-stop availability is achieved with NSSL by an automatic restart upon failures. This can be achieved by the mechanisms described in this section.

Configuring NSSL as a Generic Process (G series)

The following example SCF commands can be used to configure an NSSL HTTP server as a generic process:

```
ALLOW ALL ERRORS
ASSUME PROCESS $ZZKRN

ABORT #HTTPD
DELETE #HTTPD

ADD #HTTPD, AUTORESTART 10,                                          &
  HOMETERM $ZHOME,                                                   &
  PRIORITY 158,                                                      &
  PROGRAM $SYSTEM.COMFNSSL.NSSL,                                     &
  DEFAULTVOL $SYSTEM.COMFNSSL,                                       &
  NAME $HTTPD,                                                       &
  STARTUPMSG "HTTP; PORT 80; SUBNET $ZTC01; LOGCONSOLE *;            &
    LOGFILE HTTPLOG; HTTPBASE $DATA.OVSEC",                          &
  STARTMODE MANUAL,                                                  &
  USERID SUPER.SYSTEM ,                                              &
  CPU FIRST

START #HTTPD
INFO #HTTPD
STATUS #HTTPD
```

Before running NSSL as a generic process, we recommend that you have a working RUN NSSL command on TACL level. This command should be easy to convert to the respective SCF ADD command. For example, the NSSL startup line parameters are specified with the STARTUPMESSAGE parameter.

If running NSSL as a generic process, we recommend to send the NSSL log output to a log file instead of writing to the home terminal (default). In the example above, console logging is turned off, while log messages are written to the file HTTPLOG on the default volume.

If you want to configure multiple NSSL servers listening on the same port with Parallel Library TCP/IP or TCP/IPV6 round robin filtering, you should specify the filter key with the PTCPIPFILTERKEY configuration parameter (DEFINEs cannot be propagated to generic processes).

Please refer to the "SCF Reference Manual for the Kernel Subsystem" in the HP NonStop documentation set for further details.

Configuring NSSL as a Static Pathway Server (D series)

Either in an existing Pathway system or in a new Pathway system explicitly started for NSSL monitoring purposes you can configure an NSSL static service with the following example PATHCOM commands (in this case an NSSL HTTP server is configured):

```
RESET SERVER ASSIGN, PARAM
SET SERVER PROGRAM $SYSTEM.COMFNSSL.NSSL
SET SERVER AUTORESTART 20
SET SERVER CPUS (0:1,1:0)
SET SERVER NUMSTATIC 1
SET SERVER HOMETERM $VHS
SET SERVER OWNER SUPER.SYSTEM
SET SERVER SECURITY "O"
SET SERVER TMF OFF
SET SERVER VOLUME $SYSTEM.COMFNSSL
SET SERVER STARTUP "HTTP; PORT 80; SUBNET $ZTC01; LOGCONSOLE *;
  LOGFILE HTTPLOG; HTTPBASE $DATA.OVSEC"
SET SERVER PROCESS $HTTPD
ADD SERVER HTTPD

START SERVER HTTPD
```

Before running NSSL as a Pathway server, we recommend that you have a working RUN NSSL command on TACL level. This command should be easy to convert to the respective PATHCOM SET SERVER commands. For example, the NSSL startup line parameters are specified with the PATHCOM SET SERVER STARTUP command.

If running NSSL as a Pathway server, we recommend to send the NSSL log output to a log file instead of writing to the home terminal (default). In the example above, console logging is turned off, while log messages are written to the file HTTPLOG on the default volume specified with the SET SERVER VOLUME command.

Please refer to the "NonStop TS/MP System Management Manual" in the HP NonStop documentation set for further details.

Configuring NSSL as Multi-Homed Proxy

If NSSL is used with a proxy run mode, you can configure different TCP/IP process names for the listening and connecting sockets. One of the TCP/IP processes could even be a loopback-only process, without any connection to the network.

This "multi-homed" configuration allows to protect non-secure server ports from external access. It also allows to prevent a client proxy from being hijacked by an external attacker.

A multi-homed proxy is configured by setting the NSSL TARGETSUBNET parameter to a different TCP/IP process than the SUBNET parameter. While the SUNBET parameter determines, what TCP/IP process will listen on for incoming connections, the TARGETSUBNET parameter controls, what TCP/IP process is used for outgoing connections.

To Run NSSL as a Multi-Homed Proxy

1. Determine the TCP/IP processes the proxy should listen and connect on.
2. To start a TELNETS proxy listening on $ZTC0 and forwarding connections to a TELSERV listening on $ZTCL, port 23, issue the following command at the command prompt:

```
RUN NSSL/NAME $STN0/ TELNETS; PORT 8423; SUBNET $ZTC0;
  TARGETSUBNET $ZTCL
```

3. To start a FTPC proxy listening on $ZTCL, port 21 and forwarding FTP connections to a remote secure FTP server via $ZTC0, issue the following command:

```
RUN NSSL/NAME $FTPC/ FTPC; PORT 21; SUBNET $ZTCL;
  TARGETSUBNET $ZTC0
```

To access the FTPC proxy with the FTP client:

```
ADD DEFINE =TCPIP^PROCESS^NAME, FILE $ZTCL

FTP 127.0.0.1
```

Configuring a Loopback TCP/IP Process

If you do not want your plain servers to be available for outside connections, but force all traffic through an NSSL proxy, you may want to use a loopback only TCP/IP process. Likewise, having a client proxy listen only on a loopback TCP/IP process will prevent the client proxy from being hijacked by an external attacker.

A loopback-only TCP/IP process can be easily configured as follows:

```
TCPIP/ Name $ZTCL, NOWAIT/
SCF
> ALTER SUBNET #LOOP0, IPADDRESS 127.1
> START SUBNET #LOOP0
```

After starting the TCPIP process, you may start your servers on this process with the usual procedures.

Monitoring NSSL

Overview

NSSL writes log messages to a terminal, to a file, or to EMS. This is controlled by the parameters LOGCONSOLE, LOGFILE and LOGEMS. Log messages can be written to any combination of those three “log targets” (ie. a single one, two of them, all of them, none of them).

By default, log messages are neither written to EMS nor to a log file. This is implemented by the default values of LOGEMS, LOGFILE and LOGCONSOLE. These defaults were chosen for an easy initial setup of the NSSL object file.

Most parameters mentioned in this section can be configured both during startup as well as once NSSL is running already. In the latter case, the parameters can be changed by using the NSSLCOM command interface, please see .link to NSSLCOM section.. for more details on the usage of NSSLCOM.

What is a log message?

A log message is issued by NSSL for informational purposes, as a warning, or to indicate a fatal condition, which cannot be corrected automatically.

Why are there three different log devices?

There are three different devices which to messages can be logged, i.e. a terminal, a file, or EMS. Operators may choose their favorite location for being alerted.

For productive installation, it is recommended to either have NSSL log events to a file (LOGFILE, LOGFORMATFILE, LOGLEVELFILE) or to EMS (LOGEMS, LOGFORMATEMS, LOGLEVELEMS).

Log levels of these three devices can be different, i.e. can be written independently from each other.

What is a log level?

A log level is a number assigned to an every message in order to indicate its seriousness for the continuation of the running instance of NSSL. In general, a higher log level for a given message indicates less importance. While log levels of individual messages can not be changed, it can be controlled which levels will be displayed at all through the LOGLEVELxxx parameters.

Log Level Recommendations

The log level can be chosen individually for each log device through the parameters LOGLEVELFILE, LOGLEVELEMS and LOGCONSOLE. Depending on the device, it may be desirable to see different kind of log messages. The following table gives an indication of what “severity” individual log levels relate to:

Log Lever	**Meaning**
Level 0	fatal errors.
Up to level 10	only warnings.
Up to level 30	On Startup, NSSL issues a whole set of log messages. Those will document the current version and the settings which were used to start the NSSL process. The messages only occur once at startup.
Up to level 50	normal log messages like “close by remote client”, etc.
Up to level 89	messages only needed for trouble-shooting.
Starting from level 90	only messages to analyze extreme problems.

See the appendix for a detailed list of log messages and warnings issued by NSSL.

The following shows a sample output for the startup log messages.

```
11> run nssl telnets; port 9023
08:59:48.52|20|-------------------------------------------------------
--------
08:59:48.52|10|comForte SWAP server version
T9999G06_18Sep2003_comForte_SSLD_S40_1031
08:59:48.52|10|using openssl version 0.9.7 - see http://www.openssl.org
08:59:48.53|10|config file:  '(none)'
08:59:48.53|10|runtime args: 'TELNETS; PORT 9023'
08:59:48.53|20|--------- start settings for Logging  -----------
08:59:48.53|20| process name is $Y597
08:59:48.53|20| trace file is '*' ('*' means none)
08:59:48.54|20| max file length 20480000 bytes, length-check every 100
writes
08:59:48.54|20| console is '%' ('*' means none, '%' means home
terminal)
08:59:48.54|20| global maximum level is 9999, maximum dump length is
112
08:59:48.54|20|--------- end settings for Logging -------------
08:59:48.54|10|log level is 50
08:59:49.01|10|your system number is 12151
08:59:49.21|10|license file check OK, license file 'LICENSE',
expiration is never
08:59:49.21|30|starting collecting of random data
08:59:52.30|10|collection of 64 bytes random data finished
08:59:52.61|20|dumping configuration:
[def  ] ALLOWIP              <*>
[def  ] CACERTS              <CACERT>
[def  ] CIPHERSUITES         <0.4,0.10,0.5>
[def  ] DENYIP               <>
[def  ] LICENSE              <LICENSE>
[def  ] LOGCONSOLE           <%>
[def  ] LOGFILE              <*>
[def  ] LOGFORMAT            <76>
[def  ] LOGLEVEL             <50>
[def  ] LOGMAXDUMP           <100>
[def  ] LOGMAXFILELENGTH     <20000>
[def  ] MAXVERSION           <3.1>
[def  ] MINVERSION           <3.0>
[run  ] PORT                 <9023>
[def  ] RANDOMFEED           <64>
[def  ] SERVCERT             <SERVCERT>
[def  ] SERVKEY              <SERVKEY>
[def  ] SERVKEYPASS          <??11??>
[def  ] SLOWDOWN             <0>
[def  ] SUBNET               <$ZTC0>
[def  ] TARGETHOST           <127.0.0.1>
[run  ] TARGETPORT           <23>
[def  ] TARGETSUBNET         <$ZTC0>
[def  ] TESTWRONGDATASOCKET  <0>
08:59:52.84|50|OpenSSL cipherstring 'RC4-MD5:DES-CBC3-SHA:RC4-SHA:'
08:59:52.85|30|loading Server Certificate from file 'SERVCERT'
08:59:52.96|20|adding CA Certificate Chain Level 1/1: 'CACERT'
08:59:52.96|30|loading next Certificate Chain file from file 'CACERT'
08:59:53.07|20|Fingerprint of Root CA is
<F9E29DFC22D687C20C353BC2E37F959A>
08:59:53.07|30|loading private key from file 'SERVKEY'
08:59:53.15|10|DEFINE =TCPIP^PROCESS^NAME has value '\COMF.$ZTC0'
08:59:53.15|10|parameter SUBNET will be ignored
08:59:53.15|20|TCP/IP process is \COMF.$ZTC0
08:59:53.16|20|secure-to-plain proxy started on target host 127.0.0.1,
targetport 23, source port 902311
```

Customizing the Log Format

NSSL allows to customize the appearance of the log messages to a certain extent. For example, you may add the current date to the log message header. Please refer to the LOGFORMATCONSOLE, LOGFORMATEMS, and LOGFORMATFILE parameter descriptions for details.

To add the date to log messages

1. Start NSSL with the LOGFORMATCONSOLE parameter set to 5:

```
RUN NSSL /.../ HTTP; LOGFORMATCONSOLE 5
```

Using SHOWLOG to View a Log File

NSSL servers may be configured to write log files to disk (see parameter LOGFILE). For performance reasons, those log files are created as unstructured files:

```
15> fileinfo swaplog
$data1.comfswap
              CODE               EOF   LAST MODIFIED   OWNER RWEP   PExt
Sext
swaplog          0              5044 25sep2003 15:14 110,111 aaaa      4
28
16>$SYSTEM COMFSWAP
```

While the program is running, the log file is always open, however it may be concurrently opened for viewing. To convert the unstructured file into a readable format, a tool SHOWLOG is supplied. Invoking SHOWLOG without arguments will display a brief syntax summary:

```
20> run showlog
comForte SHOWLOG log file converter Version
T9999A05_09Feb2007_comForte_SHOWLOG_0019
usage: SHOWLOG <log file> [<process_one_line file>] [<start>] [<end>]
   <log file>     | the input log file to be converted
   <process_one_line file>  | file to write to, default is '*' meaning
the home
                              terminal
   <start>        | either byte offset from beginning OR
                    timestamp in format "ddmmmyy HH:MM:SS"
                        (example 30Jan07 21:01:59)
   <end>          | either number of bytes after beginning OR
                    timestamp in format "ddmmmyy HH:MM:SS"
                        (example 30Jan07 21:01:59)


---examples---
SHOWLOG logfile                            whole log file written to
home terminal

SHOWLOG logfile logedit 10000 1000        1000 bytes starting at offset
10000
                                                  written to EDIT file
logedit

SHOWLOG logfile * "30Jan07 20:00:10" "30Jan07 21:00:20"
                        messages in timeframe down to (ss) level to
home terminal
$DATA1 COMFSWAP 3>
```

If SHOWLOG is run with only the name of the log file as first runtime argument, it will dump the whole log file to the home terminal. The byte offset within the log file will be displayed every now and then; this allows you to limit the output of showlog to certain sections of the log file as shown below.

```
26> run showlog swaplog
comForte SHOWLOG log file converter Version
T9999A05_23Sep2003_build_0017
starting at offset 0
dumping at most -1 bytes
---processing in-file 'swaplog'
12:54:16.09|20|-------------------------------------------------------
--------
12:54:16.44|10|comForte SWAP server version
T9999G06_15Sep2003_comForte_SSLD_S40
_1031
12:54:16.48|10|using openssl version 0.9.7 - see http://www.openssl.org
12:54:16.55|10|config file:  '(none)'
12:54:16.62|10|runtime args: 'PROXYS; PORT 23456'
12:54:16.67|20|--------- start settings for Logging  -----------
12:54:16.72|20| process name is $X4J3
12:54:16.79|20| trace file is 'swaplog' ('*' means none)
12:54:16.83|20| max file length 20480000 bytes, length-check every 100
writes
12:54:16.86|20| console is '%' ('*' means none, '%' means home
terminal)
12:54:16.91|20| global maximum level is 9999, maximum dump length is
112
12:54:16.94|20|--------- end settings for Logging -------------
12:54:16.97|10|log level is 50
12:54:17.29|10|your system number is 12151
12:54:17.90|10|license file check OK, license file 'LICENSE',
expiration is never
r
12:54:17.94|30|starting collecting of random data
12:54:21.12|10|collection of 64 bytes random data finished
12:54:24.60|20|dumping configuration:


[def  ] ALLOWIP              <*>
[def  ] CACERTS              <CACERT>
[def  ] CIPHERSUITES         <0.4,0.10,0.5>
[def  ] DENYIP               <>
[def  ] LICENSE              <LICENSE>
[def  ] LOGCONSOLE           <%>
[par  ] LOGFILE              <swaplog>
[def  ] LOGFORMAT            <76>
[def  ] LOGLEVEL             <50>
[def  ] LOGMAXDUMP           <100>
[def  ] LOGMAXFILELENGTH     <20000>
[def  ] MAXVERSION           <3.1>
[def  ] MINVERSION           <3.0>
[run  ] PORT                 <23456>
[def  ] RANDOMFEED           <64>
[def  ] SERVCERT             <SERVCERT>
[def  ] SERVKEY              <SERVKEY>
[def  ] SERVKEYPASS          <??11??>
[def  ] SLOWDOWN             <0>
[def  ] SUBNET               <$ZTC0>
[def  ] TARGETHOST           <127.0.0.1>
[def  ] TARGETPORT           <23>
[def  ] TARGETSUBNET         <$ZTC0>
[def  ] TESTWRONGDATASOCKET  <0>
12:54:25.49|50|OpenSSL cipherstring 'RC4-MD5:DES-CBC3-SHA:RC4-SHA:'
12:54:25.58|30|loading Server Certificate from file 'SERVCERT'
12:54:26.79|20|adding CA Certificate Chain Level 1/1: 'CACERT'
12:54:26.86|30|loading next Certificate Chain file from file 'CACERT'
12:54:27.09|20|Fingerprint of Root CA is
<F9E29DFC22D687C20C353BC2E37F959A>
12:54:27.16|30|loading private key from file 'SERVKEY'
12:54:27.93|10|DEFINE =TCPIP^PROCESS^NAME has value '\COMF.$ZTC0'
12:54:28.00|10|parameter SUBNET will be ignored
12:54:28.06|20|TCP/IP process is \COMF.$ZTC0
12:54:28.13|20|secure-to-plain proxy started on target host 127.0.0.1,
target po
rt 23, source port 23456
```

```
---
---Byte offset is 2620
---

---
--- EOF reached, done
---
27>
```

The second runtime argument can be used to create a new EDIT file containing the log file contents. The following example shows how to convert the whole log file into an edit file (note that this can take some time for large files):

```
42> run showlog swaplog logedit
comForte SHOWLOG log file converter Version
T9999A05_23Sep2003_build_0017
starting at offset 0
dumping at most -1 bytes
writing out-file 'logedit'
---processing in-file 'swaplog'

---
--- EOF reached, done
---
43> fi logedit
$data1.tbswap
              CODE              EOF   LAST MODIFIED  OWNER  RWEP   PExt
SExt
logedit        101          5506688 18sep2003 13:11 110,110 aaaa      4
16
44>
```

The third and last runtime arguments can be used to limit the part of the file which is converted. This is helpful for the viewing large log files. The following example shows dumping a large log file. Only a limited number of log messages (totaling 10.000 bytes) after a given offset (5.000.000) are shown:

```
33> run showlog swaplog * 5000000 10000
comForte SHOWLOG log file converter Version
T9999A05_23Sep2003_build_0017
starting at offset 5000000
dumping at most 10000 bytes
---processing in-file 'swaplog'

(output not shown here)

---
---finishing dump of file before end-of-file
---

---done 34>
```

Rather than using byte offsets, SHOWLOG can also use timestamp as filters for which parts of the log file to display. The command

```
SHOWLOG logfile * "30Jan07 20:00" "30Jan07 21:00"
```

will only display log messages between the two given timestamps.

Note, that in this example by using '*' as the second runtime argument the output is written to the home terminal. When using the byte offset param or

the byte offset param and length param, the out file param must be entered as well.

Logfile/Auditfile rollover using round robin

When logging to a log file, NSSL uses round-robin to switch to a new log file. The behavior for round-robin has changed with NSSL 1046 and later and is described in this section.

Logfile rollover applies both to auditing (to the file configured with the AUDITFILE parameter) as logging (to the file configured with the LOGFILE parameter).

For all versions of NSSL, logfile rollover occurs when the logfile is greater than the size configured in the parameter LOGMAXFILELENGTH or when the audit file is greater than the size configured in the parameter AUDITMAXFILELENGTH.

Logfile rollover for NSSL versions 1045 and earlier

NSSL releases 1045 and earlier implement round-robin with two files. The last character of the file named will be replaced by a "2" for rollover, the last character can not be a "2" for the original filename.

With LOGFILE having a value of SWLOG, the current file name will always be SWLOG while the archive file name will be SWLO2.

Logfile rollover for NSSL versions 1046 and later

NSSL releases 1046 and later implement round-robin with at least 10 files. The number of files can be configured using the LOGFILERETENTION (or AUDITFILERETENTION) parameter.

Archive files created during rollover will be created by appending a number to the log file name. The number of digits of the number appended will be calculated depending on the number of files to keep.

With LOGFILERETENTION set to 10 (the default value), the archive files for a LOGFILE of SWLOG will be called SWLOG0, SWLOG1, ... SWLOG9.

With LOGFILERETENTION set to 1000, the archive files for a LOGFILE of SWLOG will be called SWLOG000, SWLOG001, ... SWLOG999.

Web Server Log

If running as a web server, the log messages describe HTTP requests received from the clients and SSL related events.

The following example shows a typical log output of a plain HTTP server.

```
$SYSTEM COMFSWAP 9> run NSSL http; port 8080
08:56:39.56|20|-------------------------------------------------------
--------
08:56:39.56|10|comForte SWAP server version
T9999G06_18Sep2003_comForte_SSLD_S40_1031
08:56:39.57|10|using openssl version 0.9.7 - see http://www.openssl.org
08:56:39.57|10|config file:  '(none)'
08:56:39.57|10|runtime args: 'HTTP;PORT 8080'
08:56:39.57|20|--------- start settings for Logging  -----------
08:56:39.57|20| process name is $Y596
08:56:39.58|20| trace file is '*' ('*' means none)
08:56:39.58|20| max file length 20480000 bytes, length-check every 100
writes
08:56:39.58|20| console is '%' ('*' means none, '%' means home
terminal)
08:56:39.58|20| global maximum level is 9999, maximum dump length is
112
08:56:39.58|20|--------- end settings for Logging -------------
08:56:39.58|10|log level is 50
08:56:39.88|10|your system number is 12151
08:56:39.90|20|dumping configuration:
[def  ] ALLOWIP              <*>
[def  ] DENYIP               <>
[def  ] HTTPBASE             <>
[def  ] HTTPZIP              <HTTPZIP>
[def  ] LOGCONSOLE           <%>
[def  ] LOGFILE              <*>
[def  ] LOGFORMAT            <76>
[def  ] LOGLEVEL             <50>
[def  ] LOGMAXDUMP           <100>
. . .
14:57:11.72|50|<10.18.24.11:33340   "GET /testmid.html HTTP/1.1"
14:57:11.76|50|>10.18.24.11:33340   200   503 "$ghs2.comfhtml.testmid"
14:57:12.16|50|<10.18.24.11:33341   "GET /butler.jpg HTTP/1.1"
14:57:12.18|50|>10.18.24.11:33341   200   5812 "$ghs2.comfhtml.butler"
14:57:12.21|50|<10.18.24.11:33342   "GET /paddle.jpg HTTP/1.1"
14:57:32.56|50|>10.18.24.11:33342   200  360139 "$ghs2.comfhtml.paddle"
14:57:44.02|50|<10.18.24.11:33347   "GET /testxxx.html HTTP/1.1"
14:57:44.03|50|>10.18.24.11:33347   404    0 "$ghs2.comfhtml.testxxx"
```

To interpret NSSL log output when running as web server

General format:

```
time|loglevel|message
```

General HTTP log message format:

```
time|loglevel|<remote_ip:port "http request"

time|loglevel|>remote_ip:port result bytes_returned "filename"
```

Line:

```
14:13:11.72|50|<10.18.24.11:33340   "GET /testmid.html HTTP/1.1"
```

Meaning:

NSSL received HTTP GET request for URL "testmid.html" which was initiated from IP address 10.18.24.11 and port 33340

Line:

```
14:13:11.76|50|>10.18.24.11:33340  200  503 "$ghs2.comfhtm.testmid"
```

Meaning:

the above request completed successfully (HTTP result 200), the returned file "$ghs2.comfhtm.testmid" has 503 bytes

Lines:

```
14:15:44.02|50|<10.18.24.11:33347   "GET /testxxx.html HTTP/1.1"
14:15:44.03|50|>10.18.24.11:33347   404  0 "$ghs2.comfhtm.testxxx"
```

Meaning:

the request failed with HTTP error 404 "file not found"

Command Interface NSSLCOM

Starting with Release S40_1031 and functionally being enhanced in later versions, NSSL is delivered with a command interface NSSLCOM. Using NSSLCOM, you can:

- get an overview of the status of NSSL
- look at the sessions which are currently open, get detailed information about single sessions (limited to certain run modes)
- view and change the following parameters (please refer to the "NSSL Parameter Reference" for the meaning of the parameters):
    - ALLOWCERTERRORS
    - ALLOWIP
    - CONTENTFILTER
    - DENYIP
    - LOGCONSOLE
    - LOGEMS
    - LOGFILE
    - LOGFORMATCONSOLE
    - LOGFORMATFILE
    - LOGFORMATEMS
    - LOGLEVELFILE
    - LOGLEVELCONSOLE
    - LOGLEVELEMS
    - LOGMEMORY
    - MAXSESSIONS
      (only in applicable run modes)
    - TRUST
      (only in run modes ending with a "C" and in run mode EXPANDS)
- execute the following additional commands
    - LOGMESSAGE
    - RELOAD CERTIFICATES
    - SSLINFO

Usage of NSSLCOM: a Sample Session

The usage of NSSLCOM is similar to the HP PATHCOM component. You connect to an existing NSSL instance using the OPEN command, then you issue commands against that instance of NSSL. The HELP command will give you a brief overview of the supported commands.

The following example session shows the following:

1. Start of NSSLCOM, connect to an NSSL instance running with the process name "$NSSL"
2. Use the STATUS command to view the current status of NSSL
3. Use the SHOW command to view the current settings of LOGLEVEL, LOGCONSOLE, LOGFILE and LOGMEMORY
4. Use the SET command to change the value of the LOGLEVEL parameter:

```
15> NSSLCOM $NSSL
GFTCOM^H16^06FEB03
OPEN $NSSL
% status
status
------------------------------------------------------------
NSSL version T9999G06_15Sep2003_comForte_SSLD_S40_1031
------------------------------------------------------------
Startup configuration:
[def  ] ALLOWIP              <*>
[def  ] CACERTS              <CACERT>
[def  ] CIPHERSUITES         <0.4,0.10,0.5>
[def  ] DELAYRECEIVE         <0>
[def  ] DENYIP               <>
[def  ] LICENSE              <LICENSE>
[par  ] LOGCONSOLE           <*>
[run  ] LOGFILE              <lproxysl>
[def  ] LOGFORMAT            <76>
[def  ] LOGLEVEL             <50>
[def  ] LOGMAXDUMP           <100>
[def  ] LOGMAXFILELENGTH     <20000>
[def  ] LOGMEMORY            <0>
[def  ] MAXVERSION           <3.1>
[def  ] MINVERSION           <3.0>
[run  ] PORT                 <32005>
[def  ] RANDOMFEED           <64>
[def  ] SERVCERT             <SERVCERT>
[def  ] SERVKEY              <SERVKEY>
[def  ] SERVKEYPASS          <??11??>
[def  ] SLOWDOWN             <0>
[def  ] SUBNET               <$ZTC0>
```

```
[def  ] TARGETHOST           <127.0.0.1>
[run  ] TARGETPORT           <65023>
[def  ] TARGETSUBNET         <$ZTC0>
[def  ] TESTWRONGDATASOCKET  <0>


------------------------------------------------------------
PROXYS mode
    active sessions right now:          3
    maximum number of active sessions:  25
------------------------------------------------------------
current heap size: 2506752
current mem pages: 115
------------------------------------------------------------
------------------------------------------------------------
Root Certificate Info:
MD5 fingerprint   <4DFF502FD33EB41911ACE1943DB3DCCA>
SHA-1 fingerprint <A71418323DDCD3140460125D3321503EB2356FE9>
------------------------------------------------------------
% show
show
LOGLEVEL     50
LOGFILE      lproxysl
LOGCONSOLE   *
LOGMEMORY    0
% set loglevel 30
set loglevel 30
log level was set to 30
% exit
exit
16>
```

Supported Commands

The following commands are supported:

- OPEN <processname>: connects to an instance of NSSL running. The process name may also be supplied as runtime parameter as shown in the example above
- HELP: lists supported commands
- STATUS: shows current status. This includes the display of the following information:
    - The startup configuration of NSSL.
    - The current configuration of NSSL. The current configuration will differ from the startup configuration when SET

commands have been used from within NSSLCOM to change values.

- In run modes ending with an "S", the fingerprint of the root certificate will be displayed.
- The number of sockets as well as the CPU ms used by NSSL will be displayed.

- SHOW: shows current values of parameters which can be altered using NSSLCOM
- SET <parameter> <value>: changes a parameter
- SSLINFO: displays the local certificate chain when NSSL is running as SSL daemon
- RELOAD CERTIFICATES: changes the server certificate chain without having to restart SWAP
- CONNECTIONS [,DETAIL]: display on overview of the current open connections of NSSL[1]
- CONNECTIONS, STATS: displays an extended usage statistics for the run modes PROXYS, PROXYC and PROXY. This statistic will yield information on how many different remote IP addresses are connecting to NSSL.
- INFO CONNECTION: displays detailed information about a single connection[1]
- RENEGOTIATE CONNECTION: forces SSL key renegotiation for a single connection[1]
- LOGMESSAGE <level> <text>: a log message with the level and text specified will be generated. This allows to test the current log settings[1].
- ROLLOVER LOGFILE: a log file rollover will be enforced regardless of the current size of the log file[1].

Multiple commands can be concatenated with semicolons in-between.

Command Reference for CONNECTION Commands

In the run modes TELNETS, PROXYS, PROXYC, MQS, MQC, ATTUNITYS, FTPS and FTPC NSSL will have a set of TCP/IP connections open during normal operation. The number of open connections can very between zero and several hundred.

Starting with NSSL release 1040, NSSL can display some information about those connections in any of the run modes above. For the run modes HTTP and HTTPS this is not possible.

CONNECTIONS

The CONNECTIONS command displays an overview of all currently open connections handled by NSSL. The following example shows the output of NSSL running in TELNETS mode with three connections handled by NSSL:

[1] These commands are not supported in all run modes of SWAP.

```
% connections
connections
| Port|--------remote connection----------|----------local connection--
-------|
| 3625|10.0.0.198:8989<--10.0.1.24:2000   |127.0.0.1:3625--
>127.0.0.1:23      |
| 3627|10.0.0.198:8989<--10.0.1.24:2010   |127.0.0.1:3627--
>127.0.0.1:23      |
| 3626|10.0.0.198:8989<--10.0.1.24:2002   |127.0.0.1:3626--
>127.0.0.1:23      |
%
```

Note: The first column contains the local port of the NonStop side of the NSSL connection. This number is used to access an individual session with the INFO CONNECTION or RENEGOTIATE CONNECTION commands.

CONNECTIONS, DETAIL

The CONNECTIONS, DETAIL command displays the list of connection, however it adds some additional information to each line. The output of the command is rather wide so it is recommended to view the output with a terminal emulator displaying 132 characters per line:

```
% connections, detail
connections, detail
|---------------remote connection-------------|---------------local connection--------------|#HS|First Handshake |Last  Handshake |
|10.0.0.198:8989<--10.0.1.24:2000             |127.0.0.1:3625-->127.0.0.1:23                |  2|05Aug04,21:26:23|05Aug04,22:38:07|
|10.0.0.198:8989<--10.0.1.24:2010             |127.0.0.1:3627-->127.0.0.1:23                |  1|05Aug04,21:27:06|05Aug04,21:27:06|
|10.0.0.198:8989<--10.0.1.24:2002             |127.0.0.1:3626-->127.0.0.1:23                |  2|05Aug04,21:26:21|05Aug04,22:34:10|
%
```

The content at the right end of the display is the abbbreviated content of the section "SSL handshake information" in the result of the INFO CONNECTION command covered in the next paragraph.

INFO CONNECTION

The INFO CONNECTION command displays detailed information about a single session as in the following example:

```
% info connection 3625
info connection 3625
accepting socket:
=================
   <Sec rem acc PROXY>[TLS_SERVER](0/1): 10.0.0.198:8989<--10.0.1.24:2000
connecting socket:
==================
   <Pln loc conn PROXY>: 127.0.0.1:3625-->127.0.0.1:23
peer certificate information:
=============================
   issuer=/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98/CN=VeriSign C

lass 1 CA Individual Subscriber-Persona Not Validated
subject=/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98/OU=Persona Not

Validated/OU=Digital ID Class 1 - Microsoft Full Service/CN=Thomas R. Burg/emailAddress=thomasburg@web.de
not_valid_before=Feb 20 00:00:00 2004 GMT
not_valid_after=Feb 19 23:59:59 2005 GMT
md5=C7D442A51F7790721E3F36C383E58DF5
SSL handshake information:
==========================
   1 SSL handshakes; First at 05Aug04,21:26:23, Last at 05Aug04,21:26:23
%
```

The command displays details about:

- Accepting socket: that’s the socket of the application which connects to NSSL. For instance in TELNETS mode, that is the connection to the remote client using SSL

- Connecting socket: that's the socket on which NSSL connects to the target application. In TELNETS mode, that is the connection to TELSERV
- Peer certificate information: if the accepting socket in TELNETS or PROXYS mode has sent a client certificate, the contents are displayed here. See section "To have NSSL require the SSL Client send a certificate" for details on enforcing client authentication.
- SSL handshake information: displays the number of SSL handshakes on the accepting socket and the timestamp of the first and last handshake.

RENEGOTIATE CONNECTION

The SSL protocol allows both parties to kick off a new SSL handshake to refresh the session keys. The RENEGOTIATE CONNECTION command lets NSSL do that from the server side. The following two log messages show that a renegotiation has been succesful.

```
22:34:08.19|50|T3|session 10.0.0.198:8989<--10.0.1.24:2002: SSL renegotiation          starting
22:34:10.35|50|T3|session 10.0.0.198:8989<--10.0.1.24:2002: cipher suite TLSv1/RC4-MD5 negotiated
```

The output of the INFO CONNECTION command will display the fact that a new handshake has happened as well:

```
%info connection 3625
info connection 3625
accepting socket:
=================
   <Sec rem acc PROXY>[TLS_SERVER](0/1): 10.0.0.198:8989<--10.0.1.24:2000
connecting socket:
==================
   <Pln loc conn PROXY>: 127.0.0.1:3625-->127.0.0.1:23
peer certificate information:
============================
   issuer=/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98/CN=VeriSign C

lass 1 CA Individual Subscriber-Persona Not Validated
subject=/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98/OU=Persona Not

Validated/OU=Digital ID Class 1 - Microsoft Full Service/CN=Thomas R. Burg/emailAddress=thomasburg@web.de
not_valid_before=Feb 20 00:00:00 2004 GMT
not_valid_after=Feb 19 23:59:59 2005 GMT
md5=C7D442A51F7790721E3F36C383E58DF5
SSL handshake information:
==========================
  2 SSL handshakes; First at 05Aug04,21:26:23, Last at 05Aug04,22:38:07
%
```

SSLINFO Command

The NSSLCOM command SSLINFO will display the local certificate chain configured through the parameters SERVCERT and CACERTS when NSSL is running as an SSL daemon.

RELOAD CERTIFICATES Command

The NSSLCOM command RELOAD CERTIFICATES allows for changing the server certificate chain without having to restart NSSL. The command has two possible syntaxes:

1. If used without an additional parameter, the command assumes the configuration parameters for the new certificate chain (SERVCERT, SERVKEY, SERVKEYPASS, CACERTS) are present in the currently configured CONFIG2 file. If no CONFIG2 file has been configured for startup, the command will fail.

2. If used with an additional parameter containing the filename of a configuration file in double quotes, the new values will be loaded from that file.

Some considerations for the command:

- The success or failure of the command will be returned to NSSLCOM. If the command fails, the prior certificate chain will remain active.
- NSSL does some limited tests on the new certificate chain. However, just as within startup of NSSL, some errors in the certificate chain cannot be detected as NSSL. It is thus recommended to immediately check the new certificate chain with the SSLINFO command as well as with connecting a new client.
- If the syntax b) of the command is used, the changes will not be permanent unless the startup configuration of NSSL is updated with the changes. It is highly recommended to always keep the certificate chain information in a CONFIG2 file and to use syntax a) as in that case the changes will be permanent without further action.

Web Server Reference

Supported MIME Types

The NSSL HTTP server will derive the mime type of a requested resource from the filename extension. The supported MIME types are:

extension	Mime type
bmp	image/bmp
cab	application/octet-stream
class	application/octet-stream
dat	text/plain
gif	image/gif
htm	text/html
html	text/html
jar	application/octet-stream
jnlp	application/octet-stream
jpg	image/jpg
txt	text/plain

If a requested resource does not match a supported extension, "application/octet-stream" is used as default.

Serving HTTP Contents

An NSSL web server satisfies browser requests by the following mechanisms:

- mapping the requested URL to a file contained in a ZIP archive.
- mapping the requested URL to a disk file

Upon receipt of a HTTP GET request NSSL will first search for the resource in the ZIP archive that was specified in the NSSL configuration. If the file is not found, NSSL will then search for a disk file matching the request.

Serving HTTP Contents from a ZIP Archive

The Guardian file system has no hierarchical structure and does only support file names 8 character long without extension. HTTP content files typically have long file names and are organized in hierarchical folders. To overcome the restrictions of the Guardian file system, NSSL can serve HTTP contents from a standard ZIP file containing the required files with their full (long) path names. Thus, the HTTP contents can be easily developed and organized on a standard workstation. For deployment with NSSL the required files simply need to be packed with a standard ZIP tool into a single archive which then is transferred to the NonStop server.

Note: NSSL does not support ZIP archives containing compressed files. Make sure to turn off compression when packing the HTTP content archive.

The ZIP archive NSSL returns files from is configured with the HTTPZIP parameter. NSSL is delivered with a default archive containing a welcome page and this manual in HTML format for on-line reading.

NSSL will map requested URLs to a zip file by matching resource specified in the HTTP GET request to the full path names of the files contained in the ZIP archive. This file name comparison is case insensitive.

Example

The URL

```
http://172.3.5.7:8080/ovweb/NewYork/mytandem.html
```

will be received by an NSSL HTTP server running on 172.3.5.7, port 8080, as the following HTTP request:

```
GET /ovweb/NewYork/mytandem.html
```

NSSL will search the ZIP archive for the file:

```
ovweb/NewYork/mytandem.html
```

Note: As NSSL will use the path information of a file in the ZIP archive, it is vital to specify the correct URL for NSSL to be able to locate it. If a given URL does not match any file WITH path information it will return error 404 (file not found). When creating a ZIP archive, please make sure that any path information you included resemble the URLs you intend to use for file access.

To Create a HTTP Contents ZIP File for NSSL

1. On your workstation, create a new directory as your HTTP content base directory.
2. Move all your required HTTP content files (HTML, GIF, JAR, CAB, ...) to this directory. If your content is organized hierarchically, create the appropriate sub-directories in the base directory.
3. To make sure that all links within your content files are correct you may want to test this locally.

4. Using your favorite zip tool, pack the files in the base directory including the subfolders into an archive. Make sure to turn off compression and to include the relative path information.
5. Using your favorite file transfer tool, transfer the ZIP archive to the NonStop system NSSL is installed on. Make sure to transfer the file in binary format.
6. Start an NSSL HTTP server with the HTTPZIP configuration parameter referring to your ZIP archive, e.g.:

```
HTTPZIP $DATA.NSSL.MYZIP
```

Mapping URLs to Disk Files

If a resource cannot be found in the HTTPZIP archive (or if no zip archive exists), NSSL will try to match the resource to a Guardian file name. This is done by removing the file name extension and the separating dot.

The location where files are searched is controlled by the HTTPBASE parameter. If set, NSSL will return files from the given $*volume.subvolume*. If the HTTPBASE parameter is omitted, no files are returned from disk.

Example

The URL

```
http://172.3.5.7:8080/mytandem.html
```

will be received by an NSSL HTTP server running on 172.3.5.7, port 8080, as the following HTTP request:

```
GET /mytandem.html
```

With the HTTPBASE parameter set to "$disk1.myhttp", the request is mapped to the NonStop Guardian file name

```
$disk1.myhttp.mytandem
```

Important Note: HTML files may be stored on the NonStop system either as EDIT files (file code 101) or as binary files (file code 0). EDIT files have a much higher processing overhead when served as HTML pages, therefore Crystal Point recommends to only store short files as EDIT files. To convert an EDIT file in a binary file, please proceed as follows:
1) Do an ASCII FTP download of the EDIT file to a PC
2) Do a Binary (!) FTP upload of the file back to the NonStop system. This will result in a file with file code 0 which is then ready to be processed by NSSL.
The same performance gain can be achieved by including the EDIT file in a HTTPZIP archive, see section "Serving HTTP contents from a ZIP archive" within this chapter.

SSL Reference

Secure Sockets Layer

The SSL (secure sockets layer) protocol is an open, non-proprietary protocol originally designed by Netscape. It has been standardized by the IETF as Transport Layer Security (TLS) protocol. SSL has been universally accepted on the World Wide Web for authenticated and encrypted communication between clients and servers and is used in millions of browsers around the world.

The History of SSL

The version numbers of SSL and TLS are somewhat confusing. Chronologically, the protocols evolved as follows:

- Netscape released SSL 2.0 with the first version of Netscape Navigator in 1994. SSL 2.0 is nearly obsolete today.
- SSL 3.0 was introduced in 1994 as an enhancement of SSL 2.0. SSL 3.0 is still used on the Internet on a daily basis, especially due to the fact that Netscape 4.7x does not support the newer protocol version TLS 1.0.
- TLS 1.0 was published by the IETF in 1999. TLS 1.0 is an enhancement of SSL 3.0 and is meant to replace it in the future. It is supported by all current browsers (Microsoft Internet Explorer 4.01 and later, Netscape Navigator 6.x, Opera 5.x). The name "TLS 1.0" is confusing as "1.0" is superior to "3.0" in this case, however that is the name the IETF people chose. TLS 1.0 may be viewed as "SSL 3.1" although there is no SSL 3.1 protocol standard.

  Summing this up, the protocols evolved as follows where the order is both chronologically and functionally "upwards": SSL 2.0 --> SSL 3.0 --> TLS 1.0.

SSL Features

The SSL protocol has the following basic properties:

- Privacy
  After an initial handshake, client and server agree on a session key which is used for a symmetric cipher algorithm to encrypt the session's payload. Example ciphers are RC4, 3-DES or AES.

- Mutual Authenticity
  Using a public-key cryptography and digital signatures, the SSL protocol allows to authenticate the server or client before exchanging confidential data.
- Session Integrity
  SSL ensures the integrity of the messages exchanged allowing client and server to verify if it has been modified by an attacker, using a Message Authentication Code (MAC). Example MAC algorithms are MD5 or SHA.

For more information on SSL we recommend the following reading:

- Stephen Thomas, "SSL and TLS essentials", Wiley Publishing 2000

Implementation Overview

Cipher Suites

NSSL uses the SSL protocol - as used in standard browsers and servers - for session security. NSSL supports SSL 2.0, SSL 3.0 and the latest version SSL 3.1, which has been standardized by the IETF as Transport Layer Security (TLS) protocol. This protocol allows for negotiating cipher suites for secure exchange of data as well as exchanging the necessary secrets at the beginning of each session in a way which is particularly strengthened against replay, insertion and man-in-the-middle attacks.

Note: we do not recommend using SSL 2.0 as it has some serious design flaws.

The selection of cipher suites is configurable, in order to make our solution customizable to the needs of individual security requirements:

- RSA certificate-based key-exchange, where the root ca is validated in the OutsideView client via fingerprint.
- Either of 3-DES, RC4 or AES as bulk-ciphers.
- Either of HMAC-SHA or HMAC-MD5 as message authentication codes.
- The actual choice of the cipher suite is at the discretion of the server and configurable.

The key lengths for symmetric encryption are:

- (Triple-DES) 3x56 = 168 bits.
- RC4 = 128 bits
- AES = 128 or 256 bits

The key lengths for message authentication are:

- (HMAC-MD5)= 128 bit
- (HMAC-SHA)= 160 bit

The cipher block chaining mode (CBC) in 3-DES guarantees the utmost security against replay/insertion as well as brute force attacks. At the current state of computer technology triple encryption is no longer a (speed) obstacle.

The authenticity of messages is granted by the 160 bit SHA hash algorithm. (HMAC-SHA) or by the 128 bit MD5 hash algorithm (HMAC-MD5).

Modulus lengths of up to 2048 bits are supported for public key values.

Auditing

An indispensable part of every security strategy is Security Auditing. The TLS protocol defines 23 Alert Messages, which may be sent or received. All these alerts are handled by NSSL, most of them are fatal. NSSL logs these alerts e.g. on the console.

Flexibility

A number of technical restrictions (available hardware encryption devices), jurisdictional restrictions (encryption unlawful in France) patent restrictions (e.g. RSA was patent protected in the US until 09/2000) and the progress in crypto-analysis might make it necessary to negotiate different cipher-suites dependent on the geographic location of the client or as part of change management. In such a case the NSSL benefits from the flexibility of the TLS protocol, which allows for negotiating the combination of cryptographic algorithms (cipher-suites) as part of the handshake. It also might be adequate to guarantee authenticity of the peer and peer data and not encrypt the session contents. The version as delivered offers a choice of TLS standard cipher suites as described above in the paragraph "cipher suite" and defined in RFC 2246. If there were any need for a change, only the algorithm would have to be implemented by Crystal Point and the changed exe and configuration files would have to be distributed, but on TLS level there would be no change.

X.509 Certificates

Certificates are a form of digital id issued by a certificate authority. A certificate authority signs a certificate with its private key, vouching for the correctness of the certificate contents. Certificates used with SSL are standardized by the X.509 specification. It is possible to built hierarchies of certification authorities, where the top level authority is called the root CA. The root CA's certificate is issued by the root CA itself; it is a so called self-signed certificate.

For SSL, the certificates are used to provide mutual authenticity. Before establishing a session, clients can authenticate a server to ensure it is connecting to a trusted site (SSL server authentication). In this case the server presents its "server certificate" along with the "certificate chain" to the client. The certificate chain is a series of certificates issued by successive CAs that reflect the certificate hierarchy up to the root certificate

Vice versa, the server can optionally request the client to present a certificate for authentication (SSL client authentication, this is currently not supported by NSSL).

NSSL supports X.509 certificates for server authentication as follows:

- If NSSL is running as SSL server (run modes HTTPS, FTPS, TELNETS, PROXYS, ATTUNITYS, MQS) NSSL will send the configured server certificates to the client. It is up to the client to check for the proper server certificates. The certificates are configured using the parameters SERVKEY, SERVKEYPASS, SERVCERT and CACERTS, please see in the parameter reference for usage of those parameters. Please see the next section on how to generate your own certificates.

- If NSSL is running as a SSL client (run modes FTPC, PROXYC, MQC), the TRUST parameter is used to configure a list of trusted root certificates. It is up to the SSL server to send the certificates, NSSL will validate the integrity of the certificate chain and check if the root certificate's fingerprint is configured in the TRUST parameter. Note that the default value * for the TRUST parameter is interpreted as "do not validate the remote certificate".

The Certificate Tools

NSSL includes several tools, which allow the generation of the required key and certificates files or to apply for a certificate. With these tools you can:

- generate a root CA certificate
- submit a Certificate Signing Request
- issue a certificate with your own root CA
- convert a BASE64 encoded certificate received from a CA to binary format
- view a certificate

The certificate tools are provided as intuitive "wizards", that will guide you step-by-step though each task. The tools can be easily accessed with your web browser by following the "Certificate Tools" link on the default NSSL HTTPS welcome page.

To invoke the Certificate Tools

1. make sure you have started an NSSL HTTP or HTTPS server with the default HTTPZIP archive included with the software.
2. point your browser to your NSSL server welcome page.
3. follow the link to the Certificate Tools.
4. select the appropriate tool that will guide you through the desired task.

The Public/Private Key Pair

Regardless of how you choose to obtain a certificate, you will need to generate a private/public key pair. The private key is stored in encrypted format protected by a pass phrase in a file complying to the PKCS#8 standard. This file is later passed to a secure NSSL process with the SERVKEY/CLIENTKEY parameter. For NSSL to be able to decrypt the private key, the password must be specified by the SERVKEYPASS/CLIENTKEYPASS parameter.

Warning: Do not give other users access to your private key! In general, private keys should be encrypted for security. The longer your pass-phrase, the better the protection of your keys.

The public key matching the private key is incorporated into the certificate along with your identification data (the server's X.509 "distinguished name").

The Certificate Signing Request

To obtain a certificate you submit your public key along with some identification data to a Certificate Authority. This so called Certificate Signing Request (CSR) is used by the CA to generate your certificate and sign it with the CA's own private key. CA's expect the CSR to adhere to a certain format. The most widely used format is specified by the PKCS#10 standard.

Obtaining a Certificate from a Third Party CA

In case you choose to obtain a certificate from an internal or external (commercial) CA you would generate a private key and a PKCS#10 CSR. You will then submit the CSR to the CA, typically by pasting it in BASE64-encoded format to the CA's web site, or sending it via email. The CA will then return the signed certificate to you, typically also in BASE64 encoded format attached to an email. The BASE64-encoded certificate can then be converted to binary certificate file, which is passed to NSSL with the SERVCERT/CLIENTCERT parameter.

NSSL needs to send the root CA certificate along with the server/client certificate to SSL clients/server for validation. Typically, the third party CA will provide their public root certificate that was used to sign the certificate. To be able to pass the root CA certificate to NSSL with the CACERTS parameter, the root CA certificate file need to be uploaded to the system you have NSSL installed on. If you received the root CA certificate in BASE64-encoded format, you may convert for NSSL usage just like a BASE64-encoded certificate.

To Submit a Certificate Signing Request for a Certificate

1. Point your browser to the Certificate Tools page.
2. Select the "Submit a Certificate Signing Request" tool.
3. When prompted to store the private key, choose a suitable name for the key file (e.g. SERVKEY.DER).
4. When prompted for the "distinguished name" to be included with the CSR, fill in the required values, e.g.

Country	USA
State or Province	WA
Locality	Bothell
Organization	Crystal Point
Organizational Unit	Development
Common Name	devserver.crystalpoint.com
Email	dev@crystalpoint.com

5. Submit the generated BASE64-encoded PKCS#10 CSR to a Certificate Authority of your choice, e.g. by pasting it to your CA's web site or attaching it to an Email.
6. Wait for the CA to return the certificate generated from your CSR.

To Convert the BASE64-Encoded Certificate Received from a CA

1. Point your browser to the Certificate Tools page.
2. Select the "Receiving a Certificate" tool.
3. When prompted for the certificate, paste the BASE64-encoded certificate received from your CA into the form.
4. When prompted to store the certificate, choose a suitable name for the certificate file (e.g. SERVCERT.DER). You may want to store the certificate in the same directory where you stored your private key file.

Acting as Your Own CA

If you choose to issue a certificate as your own CA, you would need to generate a root CA certificate and private key. The root CA certificate is a "self-signed" certificate as it is signed with the root CA's own private key.

Warning: Do not give other users access to your root CA private key! If this key is compromised, malicious users can create certificates that will appear to be signed by your CA certificate. In general, private keys should be encrypted for security. The longer your pass-phrase, the better the protection of your keys. The root CA's private key should also be stored at a secure place. For example, you could store it on a removable disk that you can lock away.

Using the root CA private key and certificate you would then generate a certificate from a previously created CSR. In other words, you would perform the same task as a third party CA.

To Issue a Certificate

1. Point your browser to the Certificate Tools page. If you have already generated a root CA certificate you may continue with step 7.
2. Select the "Generate a root CA Certificate" tool.
3. When prompted to store the private key, choose a suitable file name for your root CA key (e.g. CAKEY.DER). You may prefer to store the root CA key file on a removable media for utmost security.
4. When prompted for the "distinguished name" to be included with the root certificate, fill in the required values, e.g.

Country	USA
State or Province	WA
Locality	Bothell
Organization	Crystal Point
Organizational Unit	Development

Common Name	devca.crystalpoint.com
Email	dev@crystalpoint.com
Serial No	1
Valid Until	01/01/2010

5 When prompted to store the generated root certificate, choose a suitable file name (e.g. CACERT.DER).

6 After finishing the generation of s root CA certificate select the "Submit a Certificate Signing Request" tool.

7 When prompted to store the private key, choose a suitable name for the key file (e.g. SERVKEY.DER).

8 When prompted for the "distinguished name" to be included with the CSR, fill in the required values, e.g.

Country	USA
State or Province	WA
Locality	Bothell
Organization	Crystal Point
Organizational Unit	Development
Common Name	devserver.crystalpoint.com
Email	dev@crystalpoint.com

9 After the CSR is generated, continue with the "Issue a Certificate" tool. The form expecting a BASE64-encoded CSR will automatically show the CSR you had created before.

10 When prompted to load the root CA's private key and certificate, enter the filenames of root CA files that you the previously stored on your local system. Don't forget to enter the password for your CA private key file.

11 After you have successfully generated the certificate from the CSR, continue with the "Receiving a certificate" tool. The form expecting a BASE64-encoded certificate will automatically show the certificate you have created before.

12 When prompted to store the certificate, choose a suitable name for the certificate file (e.g. SERVCERT.DER). You may want to store the certificate in the same directory where you stored your private key file.

Configuring SSL for Production Running as SSL Server

The default installation of NSSL is streamlined to enable an easy setup and immediate testing. NSSL is delivered with a set of certificate and key files which can be used out-of-the-box for testing and evaluation purposes.

For a secure production installation, it is recommended to configure NSSL to use your **own** certificate and key files. Using the default files and settings for a production installation may compromise the security of the system.

This section will describe how to generate your own certificates using the Certificate Tools supplied with NSSL. It also explains how NSSL is

configured to use these certificates for a production installation. For a more detailed explanation about the concept of certificates, see the section "X.509 Certificates" of this chapter.

Using Your Own Server Key and Certificate Files

For a production installation, it is required to use your own keys and certificates. You will need at least the following components to configure SSL Server Authentication with your own production certificates:

1. a private key (protected by a pass phrase)
2. a server certificate incorporating the public key matching the private key
3. the certificate of the root CA that issued (i.e. signed) the server certificate

To obtain the certificates required for SSL server authentication you may choose one of the following options:

- Purchase a server certificate from a commercial CA
- Obtain a server certificate from an existing internal Certificate Authority of your organization.
- Be your own (root) Certificate Authority to issue a server certificate.

Which option you choose for your production system depends on the nature of your application, the type of users accessing it and on the existing security infrastructure.

If your organization already maintains an internal public key infrastructure (PKI), you would want to obtain a server certificate from an internal CA.

If your server is accessed by external internet users (e.g. customers) that do not know your organization yet, you would probably purchase a server certificate. Remember certificates are used to establish trust. The users trust the CA you purchased your server certificate from, while the CA vouches for your certificate's correctness.

Today's browsers (such as Microsoft Internet Explorer or Netscape Navigator) are delivered with the root certificates of commercial CAs such as Verisign or Thawte, which are "trusted" automatically if an SSL connection is made. Thus, if your server certificate is signed by a "trusted" CA, the browser will **not** display a warning message that the server certificate cannot be validated. For an internal application, this kind of automatic trust may not be desirable. In fact, using a commercial root CA could even present a security hole. If, for example, an OutsideView SSL connection to an NSSL TELNETS proxy is authenticated with a commercial root CA, a man-in-the-middle attack could be launched using another server certificate purchased from the same CA.

If you want to secure access to an application for internal users only, you would probably prefer using your own root CA to issue the server certificate. As your users know your organization already, they can choose to trust your root CA that issued the server certificate. If the users access NSSL with their browser, they will be able to look at the certificates, which will allow them to validate if they are really originating from a trusted source (your organization), for example by checking the certificate's fingerprint. They could then add your root CA certificate to their browser's list of trusted CAs.

TELNETS clients would also authenticate your NSSL TELNETS proxy using your root CA's fingerprint. In case of OutsideView, you would specify your root CA's fingerprint in the HTML used to deliver the applets. This would prevent man-in-the-middle attacks as you have full control over the server certificate generation.

Starting NSSL for Production as SSL Server

After following the instructions in the previous sections to obtain a server certificate, you should have the following files on your local system:

- the server private key file (e.g. SERVKEY.DER)
- the server certificate (e.g. SERVCERT.DER)
- the root CA certificate(s) that were used to sign the server certificate. This could be a single certificate (e.g. CACERT.DER) or a certificate chain consisting of multiple certificates (e.g. VERIROOT.DER and VERIINT.DER).

As these files are required for NSSL startup, you will need to upload them to the NonStop system NSSL is installed on.

Once the files have been transferred to the NonStop system, NSSL can be started to use you production keys and certificates by specifying the appropriate parameters.

To Have NSSL Require the SSL Client Send a Certificate

NSSL supports client authentication when running in SSL server mode (PROXYS, FTPS, TELNETS, HTTPS, MQS, ATTUNITYS). The behavior is controlled by the TRUST parameter (please note: the parameter has different meanings for NSSL running in server or client mode).

TRUST set to "*" (default) will disable the checking, thus no client cert will be required.

When TRUST contains a certificate filename this certificate will be sent to the client. The client will send back a certificate signed by the one sent to it. If the client sends no certificate or an invalid one, the connection will be rejected.

To Start NSSL with Your Own Certificate and Private Key

1. Using your favorite file transfer program, transfer the following files to your NonStop system:
   - the server private key file (e.g. SERVKEY.DER)
   - the server certificate (e.g. SERVCERT.DER)
   - the root CA certificate(s) (e.g. CACERT.DER or VERIROOT.DER and VERIINT.DER).

   Transfer the files in binary mode, stripping the filename extension. You may want to place the files into a separate GUARDIAN subvolume (e.g. $SYSTEM.MYCERT)
2. At the command prompt, issue the following command:

```
RUN NSSL/NAME $HTTPS/ HTTPS; SUBNET $ZTC0; PORT 8443;
   SERVKEY $SYSTEM.MYCERT.SERVKEY; SERVKEYPASS mysecret
   SERVCERT $SYSTEM.MYCERT.SERVCERT; CACERTS $SYSTEM.MYCERT.CACERT
```

where

- the keyword "HTTPS" designates the NSSL run mode as a secure web server.
- the parameter "SUBNET" specifies the TCP/IP process NSSL should run on. You may omit this parameter, in which case NSSL will assume $ZTC0 as default.
- the parameter "PORT" reflects the port number NSSL should listen on for HTTPS connections. Note, that to start an NSSL secure web server on the well known HTTPS port (443), SUPER group rights will be required.
- the parameter "SERVKEY" points to your server private key file.
- the parameter "SERVKEYPASS" specifies the pass phrase you used for private key encryption.
- the parameter "SERVCERT" points to your server certificate file.
- the parameter "CACERT" points to the certificate file(s) of the root CA that issued your server certificate..

3 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

Configuring SSL for Production as SSL Client

In run modes PROXYC and MQC NSSL will be an SSL client. This section only is relevant for those run modes.

The default installation of NSSL is streamlined to enable an easy setup and immediate testing and will not verify the remote certificate for authenticity.

For a secure production installation, it is recommended to configure NSSL to verify the remote certificates using the TRUST parameter. Not doing so may compromise the security of the system.

To Start NSSL with Verification of the Remote Certificate

1 Determine the MD5 fingerprint of the root certificate of the remote systems you want to communicate with. As an example, we use the following MD5 fingerprint:

```
F9E29DFC22D687C20C353BC2E37F959A
```

2 Configure the fingerprint in the TRUST parameter such as in the following startup command:

```
RUN NSSL/NAME $FTPC/ FTPC; &
   TRUST F9E29DFC22D687C20C353BC2E37F959A
```

3 NSSL will now start with the parameters specified on the command line. It will output initialization messages to your terminal. Please check these messages for any errors.

4 If you have multiple systems you want to communicate with which have different fingerprints, you can enter a list of comma-separated values for the TRUST parameter.

To Have NSSL Send a certificate to the SSL Server

If Client Authentication is required NSSL can send a client certificate or a client certificate chain to the server.

Comment: See chapter "The Certificate Tools" or the previous chapter "Configuring SSL For Production Running As SSL Server" to find out more about how to configure certificates with NSSL.

The parameters could be set to cover 3 scenarios:

1 If CACERTS and CLIENTCERT are set to '*', NSSL will sent NO certificate to the server (this is the default setting).

```
RUN NSSL/NAME $PROC/ PROXYC; …..;
   CACERTS *; CLIENTCERT *; ….
```

2 To send a self-signed certificate to the server CACERTS has to be set to '*' and CLIENTCERT/CLIENTKEY/CLIENTKEYPASS must point to a valid self-signed certificate.

```
RUN NSSL/NAME $PROC/ PROXYC; …..;
   CACERTS *;
   CLIENTKEY $SYSTEM.MYCERT.CLNTKEY; CLIENTKEYPASS mysecret
   CLIENTCERT $SYSTEM.MYCERT.CLNTCERT;
```

3 If CACERTS contain the signing certificate(s) NSSL will sent the whole certificate chain to the server.

```
RUN NSSL/NAME $PROC/ PROXYC; …..;
   CACERTS $SYSTEM.MYCERT.CACERT;
   CLIENTKEY $SYSTEM.MYCERT.CLNTKEY; CLIENTKEYPASS mysecret
   CLIENTCERT $SYSTEM.MYCERT.CLNTCERT;
```

TLS Alerts

If a TLS Alert happens on an SSL-encrypted session, the TLS alert number will be logged. The following message is an example for a log message of this type: a plain Telnet client tried to connect on the encrypted socket, resulting in a TLS alert "50" (DecodeError).

```
13:37:18.53|30|TLS Alert: 50
```

The following table contains the TLS alert numbers for TLS 1.0. For more information about the individual alerts, please refer to the TLS specification RFC 2246 (available under http://www.ietf.org).

TLS Alert Number	TLS Alert name
0	close_notify
10	unexpected_message
20	bad_record_mac
21	decryption_failed
22	record_overflow
30	decompression_failure
40	handshake_failure
42	bad_certificate
43	unsupported_certificate
44	certificate_revoked
45	certificate_expired
46	certificate_unknown
47	illegal_parameter
48	unknown_ca
49	access_denied
50	decode_error
51	decrypt_error,
60	export_restriction
70	protocol_version
71	insufficient_security
80	internal_error
90	user_canceled
100	no_renegotiation

Performance Considerations

Introduction

"There is no such thing as a free lunch" – using NSSL to encrypt Telnet, FTP or other traffic will consume some CPU cycles on your NonStop host. The natural question "how many CPU does it cost to run encryption" has no simple answer, it will depend on many factors:

- In general:
  - how many SSL connections are created – the initial setup of an SSL session involves several public-keys operations, which require some CPU intensive calculations.
  - the key sizes used for the public/private key – using a more secure 1024 bit key pair will cause more overhead for the initial setup than a 512 bit RSA key pair.
  - the selected cipher for bulk encryption – for example a cipher using 168 bit 3DES will consume more CPU cycles than a 128 bit RC4 based cipher suite.
- For Telnet traffic:
  - the number of concurrent sessions – this simply means how many parallel Telnet sessions are running.
  - the throughput on each individual session – a TACL waiting quietly for user input will consume no CPU at all where as a FILEINFO $*.*.* will result in a lot of traffic across your network which needs to be encrypted.
  - the nature of the host application – a lot of small transfers from and to the terminal will generate more SSL protocol overhead than a few large transfers.
- For HTTP traffic:
  - number of HTTP requests – how many people are downloading HTML content.
  - type of the individual request – is it a download of a short Web page or a download of a huge Java applet.
- For FTP and MQ traffic:
  - the size of the transmitted data.
- For EXPAND traffic:
  - the volume and packet size of the transmitted data

So basically there is no general answer to the question, it will depend on your individual system use.

However, some extensive measurements using MEASURE for Telnet traffic showed that today's NonStop systems aren't as bad in number crunching (and that's what encrypting and decrypting is basically about) as one would think.

The following sections will show the results of some selected measurements. The conclusions drawn from these can be used to estimate what performance behavior you can expect on your system.

Note: Unless noted otherwise, all measurements referred to in this chapter have been performed on a 2 processor S7600 system with SSL_RSA_WITH_RC4_128_SHA as selected cipher suite. As of this writing, no measurements have been made for HTTP traffic. However, we feel that HTTP traffic is of less relevance than Telnet traffic, because by its nature HTTP traffic is less repetitive than terminal traffic through.

Performance Analysis of SSL Session Establishment

The performance impact of the initial SSL session setup should be viewed separately. As explained before, establishing an SSL session involves several CPU intensive public key operations. The amount of CPU cycles consumed is depending on the key sizes used.

The following table shows the CPU consumption of an SSL session connect (without any data transfer taking place) for the various key sizes (measured on a S7600):

Key size [bits]	Approximate CPU consumption on S7600 [milliseconds]
512	43
1024	120
2048	581

The exponential increase of the CPU consumption with the key size is related to the nature of the RSA public key operations. Remember, "there is no such thing as a free lunch". You should carefully analyze what key size provides a reasonable protection for your application. The selected key size should be reflecting the value of the key for potential attackers.

It is very hard to predict future developments both in cryptography and computer technology which makes it next to impossible to tell in advance what key size will be sufficient in the years to come. We recommend using a key size of 1024 bits for the time being.

Performance analysis of SSL FTP traffic

To get an indication of the performance of the NSSL when acting as an FTP proxy the average transfer rate and CPU consumption has been measured while a file with 50 MB of data has been transferred SSL encrypted several times via the NSSL. The measurement was done when downloading that file from a NonStop system onto a PC (FTPS)

The following table shows the result of the measurement (transferring files of 50MB size, using NSSL 1036):

Mode	Time elapsed [s]	CPU time used [s]	Through-put [KB/s]	CPU ms/MB transf.	CPU usage
Unencrypted (FTPSERV only)	32	n/a	1526	n/a	n/a
RC4-MD5	47,99	11,826	1017	248	25%
RC4-SHA	48,16	12,838	1014	269	27%
DES-CBC3-SHA	52,04	34,938	938	733	67%
AES128-SHA	48,86	16,58	999	348	34%

Please bear in mind that the measured transfer rate does not only depend on the performance of the NSSL but also on the network throughput and the performance of the remote FTP client or server.

Performance Analysis of SSL EXPAND Traffic

The performance impact of adding SSL encryption to EXPAND over IP traffic will depend on both throughput and individual packet sizes.

The following table shows some results measured on an S86.000 system with two Pathway server exchanging messages using EXPAND over IP. The table lists the following values with and without usage of SSL using the cipher suite RC4-MD5:

- Maximal throughput in Kilobyte per second
- CPU busy value under this throughput
- CPU cost of all involved processes in milliseconds per Megabyte transferred

Msg Size			without SSL			with SSL			
Send	Reply	Through-put (KByte/s)	Total CPU cost (ms/MB)	CPU Busy	Through-put (KByte/s)	Total CPU cost(ms/MB)	CPU Busy	CPU usage increase with SSL (ms/MB)	Through-put with SSL
0	0	0,0	n/a	26,4%	0,0	n/a	45,3%	n/a	n/a
64	64	129,1	2056	25,3%	55,0	8544	44,8%	6488	42,6%
128	128	254,5	1032	25,1%	108,7	4313	44,7%	3281	42,7%
256	256	496,9	520	24,7%	210,6	2191	44,0%	1671	42,4%
512	512	899,0	266	22,8%	393,5	1132	42,5%	865	43,8%
1024	1024	1498,8	147	21,0%	694,6	608	40,3%	461	46,3%
2048	2048	2210,9	95	20,0%	1179,8	362	40,7%	267	53,4%
4096	4096	3398,1	47	15,2%	1819,4	251	43,6%	205	53,5%
8192	8192	4916,5	54	25,2%	2560,4	190	46,3%	136	52,1%

16384	16384	6481,8	47	29,3%	1467,3	193	27,0%	145	22,6%
28000	28000	7458,3	45	31,7%	2133,0	194	39,4%	149	28,6%
56000	56000	8404,1	42	33,9%	3843,0	177	65,0%	135	45,7%

Using this table together with MEASURE data of your current expand traffic, it is possible to estimate the additional CPU usage due to encryption.

Summary

There is no answer to the seemingly simple question: "How much CPU cycles will 128 bit encryption burn on my system ?". To understand why, consider asking an automobile expert the question, "How much fuel will I need for my vacation trip ?" (without giving away more information). Regardless how much the expert knows about cars and engines, he will not be able to give an answer unless you tell him:

- the make of the car
- where you want to go
- your driving habits

Experience shows that with Telnet, only the SSL session setup is relevant for CPU consumption whereas for run modes in which bulk data transfer takes place (such as encrypting FTP or MQ traffic) both session setup and bulk transfer may be relevant. The tables listed above should help you in estimating CPU load.

Troubleshooting

Troubles with the Browser

Browser unable to connect

If your browser cannot connect to the NSSL HTTP(S) server, please check the following:

- Did NSSL start successfully? Check with SCF STATUS PROCESS for a TCP LISTEN on the port you specified for NSSL.
- Did you specify the correct URL?

  If NSSL has been started with a port other that the well known HTTP or HTTPS port, make sure to specify the port in the URL, e.g.

```
http://10.2.3.4:8080/
```

  or

```
https://10.2.3.4:8443/
```

Browser displaying garbage page

When connecting to an NSSL HTTPS server with a browser, check if the requested URL specifies the correct protocol (https://...).

Connection closed by NSSL immediately after setting-up a secure connection

If the connection is closed immediately after setting-up a secure connection, you might have forgotten to specify "https" as protocol in your URL.

HTTP 404 – File not found

This error indicates that NSSL did not find the requested resource. Please check if the requested resource is present on the specified HTTBASE subvolume or HTTPZIP archive. If you created your zip file with path information, make sure that you have specified the path in the resource URL.

Troubles with NSSL

Address already in use

If the message "Fatal error: Could not listen on socket: Address already in use" appears, please check whether the Source Port, which you assigned as PORT parameter is not in use by any other process.

Could not open xxx file

If the message "Could not open xxx file" appears, please check whether the file with the specified name (e.g. key file , certificate file, log file) is in use by any other process.

Decode error

If a message with a "Decode Error" occurs in the NSSL log, a client may have tried to create a non-secure connection to a secure NSSL server (HTTPS, TELNETS, etc.). When connecting to an NSSL HTTPS server with a browser, check if the requested URL specifies the correct protocol (https://...).

Handshake error

If a message with a "handshake error" occurs in the NSSL log, please check the following:

- does the client support the configured SSL protocol versions (MINVERSION, MAXVERSION). For browsers it may be necessary to explicitly enable TLS if MINVERSION is set to 3.1-
- does the client support the configured CIPHERSUITES? For browsers, it may be necessary to upgrade to a "strong encryption" version, as weak 40 bit encryption ciphers are not supported by NSSL.

Invalid address

If the message "Invalid address..." appears, please check whether PARAMS TARGETHOST and TARGETPORT describe a valid host::port address in your network.

Problem with checking license file

If you see a message like the following during startup:

```
09:15:30.77|20|-----------------------------------------------------
--------09:15:30.77|10|comForte NSSL server version
T9999G06_19Sep2003_comForte_SSLD_S40_103109:15:30.78|10|using openssl
version 0.9.7 - see http://www.openssl.org
09:15:30.78|10|config file:  '(none)'
09:15:30.78|10|runtime args: 'TELNETS; PORT 9023; TARGETPORT 6502'
09:15:30.78|20|--------- start settings for Logging  -----------
09:15:30.79|20| process name is $Y59A
09:15:30.79|20| trace file is '*' ('*' means none)
09:15:30.79|20| max file length 20480000 bytes, length-check every 100
writes
09:15:30.79|20| console is '%' ('*' means none, '%' means home
terminal)
09:15:30.79|20| global maximum level is 9999, maximum dump length is
112
09:15:30.79|20|--------- end settings for Logging -------------
09:15:30.80|10|log level is 50
09:15:31.16|10|your system number is 12151
 comForte SWAP server version T9999G06_19Sep2003_comForte_SSLD_S40_1031
 --- Fatal Error:

 problem with checking license file:
 system number conflict: license file LICENSE has 37936, system has
12151

 --- aborting.
```

you have some problems with your license file. For the run modes HTTPS, TELNETS, PROXYC, PROXYS, MQS, MQC, FTPC and FTPS you need a license file from Crystal Point which allows the usage of that run mode. The license file is tied to your system number.

Security violation (error 4013)

If NSSL fails with a security violation, you may have attempted to start NSSL to listen on a PORT smaller than 1024 without having a SUPER group user id.

Excerpt from the "Tandem TCP/IP programming manual":

EACCES (4013)

Cause. A call to bind or bind_nw specified an address or port number that cannot be assigned to a non-privileged user. Only applications whose process access ID is in the SUPER group (user ID 255,n) can bind a socket to a well-known port.

Effect. The bind or bind_nw call failed.

Recovery. Specify another port number or address, or rerun the application with a process access ID in the SUPER group (user ID 255,n).

Appendix

NSSL Log Messages and Warnings

This section lists all log and warning messages issued by NSSL.

Startup Messages

This section contains messages which are displayed during startup and which are of an informational nature only.

comForte SWAP server version <NSSL version number> version_info

Appears right after startup and notifies about the version umber of the NSSL

using openssl version 0.9.7 - see http://www.openssl.org);

Notifies about the OpenSSL version bound to the NSSL

config file: '<filename>

Displays the name of the configuration file the NSSL has been started with

runtime args: '<list of runtime args>

If the NSSL has been started with runtime args instead or in addition to the a config file or TACL params those args are being displayed.

log level is <log level>

Informs about the log level the NSSL has been started with

your system number is <system number>

The system number of the NSK system on which the NSSL has been started. This number must be the same as the system number given in the license file.

license file check OK, <name of license file>

Notifies about the successful license check. Default license file name in LICENSE

starting collecting of random data

Notifies about the process of collecting random data.

collection of <number> bytes random data finished

Informs about completion of collecting random data. <number> is the number of collected bytes.

no HTTPZIP archive configured

Informs that the NSSL has been started without a HTTPZIP archive being configured. The HTTPZIP archive is a means to allow the NSSL although running (as a Web server) under Guardian to support Unix like directory structures i.e. real URLs..

couldn't open HTTPZIP archive <filename>, error: <error number>

Warning to inform that the HTTPZIP archive identified by filename could not be opened for the reasons specified by error number. Error number is a GUARDIAN file system error.

no HTTPZIP archive will be used

This message occurs in cases were no HTTPZIP archive has been configured for the NSSL. In this case all resources requested by the browser must reside in the NSK subvolume where the NSSL resides or in the NSK subvolume given by the param HTTPBASE.

using HTTP ZIP archive <filename>

Displays the filename of the HTTPZIP archive.

sysnum wild card not allowed with unlimited expiration date

The license check failed. Please contact Crystal Point.

sysnum wild card not allowed with expiration date more than 6 months in advance

The license check failed. Please contact Crystal Point.

DEFINE =TCPIP^PROCESS^NAME has value '%s'"

Notification about which TCP/IP process name is being used

parameter SUBNET will be ignored

Notification that the value of the SUBNET parameter will be overridden by the DEFINE =TCPIP^PROCESS^^NAME

TCP/IP process is <process name>

Notification about the TCP/IP process used for the communication in the current context of this message.

parameter SUBNET was evaluated

Notification that a SUBNET param has been found and will be used to determine the TCP/IP process

starting HTTPS server on port <port number>, default dir <directory name>

Notification that the NSSL is starting in HTTPS mode listening on port given by port number and defaulting to the subvolume given by the param HTTPBASE. If the latter param ist not set the default dir is the one where the NSSL resides and <default directory> in this message is left empty.

plain HTTP server on port <port>, default dir <directory name>

Notification that the NSSL has been started as a Web server in plain HTTP mode listening on port given by port number and defaulting to the subvolume given by the param HTTPBASE. If the latter param ist not set the default dir is the one where the NSSL resides and <default directory> in this message is left empty.

plain-to-plain proxy started on target host <hostname or ip address>, target port <port number>, source port <port number>

Notification that the NSSL has been started in plain-to-plain mode (i.e. no encryption takes place on neither side). Target host, target port and source port involved in the plain-to-plain session are given in the message.

secure-to-plain proxy started on target host <hostname or ip address>, target port <port number>, source port <port number>

Notification that the NSSL has been started in a mode accepting connections for secure data and connecting to the target host on target port for plain data.

plain-to-secure proxy started on target host <hostname or ip address>, target port <port number>, source port <port number>

Notification that the NSSL has been started in a mode accepting connections for plain data and connecting to the target host on target port for secure data.

FTP server proxy started on target host <hostname or ip address>, target port <port number>, source port <port number>

Notification about the NSSL beint started in FTPS mode and connecting to target host, on target port while accepting connections on source port.

FTP client proxy started on source port <port number>

Notification about the NSSL beint started in FTPC mode and accepting connections on source port.

dumping configuration: <config setting>

Displays the settings of the configuration params

loading Server Certificate from file <filename>

Notification that the server certificate is being loaded from the file given by filename

loading next Certificate Chain file from file <filename>

Notification that the next of a sequence (chain) of signing certificates are being loaded by the NSSL.

Fingerprint of Root CA is <MD5 fingerprint>

Notification about the MD5 calculated fingerprint of the root certificate. If fingerprint checking is activated on the client side, this fingerprint must be preconfigured there.

loading private key from file <filename>

Notification about the server private key (see param SERVKEY) being loaded from the file identified by filename.

adding CA Certificate Chain Level <curr number>/<max number> <filename>

Notification about how many certificates the CA certificate chain contains, which of these certificates are currently processed and what the filename of this is.

Connection closed by remote client

This log message will be issued any time a remote client disconnects unexpectedly. In most cases (especially when running in TELNETS mode), this log message can be safely ignored.

Warning Messages

The following messages are displayed under conditions where NSSL can recover from an error and will continue to run.

Firewall: connection rejected from: <ip address>

Warning about a connection from a remote host identified by ip address being rejected. This message may appear in conjunction with the ip filtering function of the NSSL, see parameters ALLOWIP and DENYIP for details

invalid request <request-error> from <ip address>

Warning about the receiving of an invalid HTTP request received from remote host identified by ip address. The parameter <request-error> will display a detailed error message.

certificate not yet valid

Warning that the certificate being currently processed by the NSSL is not yet valid, i.e. has a "from date" starting in the future.

certificate expired

Warning that the certificate being currently processed by the NSSL has expired i.e. is not valid any longer.

request too big for buffer, <number> bytes read without EOR

Warning about an HTTP request with an invalid request length. Watch for other error messages which come along with this warning.

<ip address> - received HTTPS request in HTTP mode

Notification about receipt of a HTTPS request received from ip address when running in HTTP mode

F|#<session>-<ip address> login without SSL rejected

[FTPS mode only] The remote client identified by ip address attempted an unsecured login on a secured port. The login was rejected.

TLS Alert: <TLS alert number>

Warning about an TLS alert received within current session

TLS Exception

Warning about a TLS exception received in current TLS session. Watch for message which come along with this and which give additional information.

remote fingerprint <fingerprint> rejected

Warning that a certificate received from the remote SSL server was rejected because the MD5 fingerprint did not match the one configured with NSSL param TRUST. The calculated fingerprint is displayed.

OnAccept1Complete: error <error number>

Internal error occurred during acceptance of a TCP/IP connection. Watch for other message which come along with this one in order to decide whether and what action has to be taken.

OnAccept2Complete: error <error number>

Internal error occurred during acceptance of a TCP/IP connection. Watch for other message which come along with this one in order to decide whether and what action has to be taken.

F<session>|<-- unexpected reply to PASV command from FTP server: <reply>

Warning about receipt of an unexpected reply from the remote FTP server upon requesting Passive Mode FTP. Check whether the remote FTP server supports passive mode FTP.

F<session>|<-- reply to STOR/RETR/LIST command from FTP server has error: <reply>

Warning that the reply from the remote FTP server upon one of the mentioned requests is erroneous. If this happens frequently contact your support representative.

F<session>|<-- reply to PORT command from FTP server has error: <reply>

Warning about the receipt of an erroneous reply from a FTP server upon requesting active mode FTP. Check whether the remote FTP server supports active mode FTP.

Informational Messages

The following messages display information about actual events. No corrective action is necessary.

issuer= <an certificate issuer>

Notification about the issuer of the currently processed certificate during SSL session establishment

F|#<session>-<ip address> close

[FTPS mode only] Notification about a session with host identified by ip address being closed.

F|#<session> - <ip address> new connection

[FTPS mode only] Notification about a new connection being established with remote client identified by ip address

Fatal Errors

The following messages are displayed in situations where a fatal error occurred. NSSL will abend because it can not recover from that error.

Fatal Error: could not listen on port <port number>, error <error number >

Error condition which is caused either by another application listening on same port or by configuring the NSSL with a PORT param less then 1024 while not starting the NSSL under the SUPER user logon. The NSSL terminates.

Fatal Error: AWAITIOX file <filename> (filenum <filenumber>) completed with error <error number>

A nowaited operation on file identified by filename completed with a filesystem error. Watch for other message which come along with this one in order to decide whether and what action has to be taken.

Fatal Error: fatal error in proxy server, OnAccept: Accept failed

Internal error condition. Receiving a connection failed. Watch for other message which come along with this one in order to decide whether and what action has to be taken.

Fatal Error: HTTP fatal accept error in HTTP server

Internal error condition. Receiving a connection failed. Watch for other message which come along with this one in order to decide whether and what action has to be taken.

out of memory in CHTTPServer::OnAccept

Internal error condition. Contact your support representative.

Fatal SSL error <error text>, exiting

A fatal error specified by error text has occurred. The NSSL is being ended. Please consult your support representative for further action.

Can't open input file <filename>

Warning that the file given by filename cannot be opened. The severity of this message depends on the context in which it appears. If for example the server key file cannot be opened the NSSL terminates. Watch for other messages surrounding this one.

Fatal SSL error <error number> , exiting

A fatal error during SSL processing has occurred which causes the NSSL to terminate. Watch for earlier message which give additional information.

illegal parameter <param value> for MINVERSION needs to be one of 2.0/3.0/3.1

Warning about an invalid setting of param MINVERSION. The allowed settings are being displayed with the message. This message appears during start up of the NSSL. The NSSL will not start.

illegal parameter '%s' for MAXVERSION needs to be one of 2.0/3.0/3.1

Error message about an invalid setting of param MAXVERSION. The allowed settings are being displayed with the message. This message appears during start up of the NSSL. The NSSL will not start.

MAXVERSION can not be smaller than MINVERSION

Warning that an invalid param setting have been detected where MINVERSION is smaller than MAXVERSION. This message appears during start up of the NSSL. The NSSL will not start.

Index

S

T

U

W

X