SSFT explained

The official

Secure Socket File Transfer

documentation

Jarle (jgaa) Aase
Jgaa's Internet

jgaa@jgaa.com

SSFT explained: The official Secure Socket File Transfer documentation

by Jarle (jgaa) Aase

Published v0.01.01 2001/10/19

This manual is produced using DocBook (http://www.docbook.org/). The original manuscript is included in the ssft source code distribution. You can browse the on-line version at ssft.jgaa.com (http://ssft.jgaa.com/) or download the Adobe pdf version (http://ssft.jgaa.com/ssft-manual.pdf).

Table of Contents

List of Tables

List of Examples

About this Manual

1. Purpose / Scope of this Document

This is the official documentation on *the Secure Socket File Transfer*. It is primarily targeted against system administrators.

2. Copyrights and Trademarks

Chapter 1. Introduction to SSFT

1.1. Background

SSFT is a utility designed for system-administrators to copy files securely between servers over insecure networks (like the Internet) or between different operating systems on a non-trusted LAN. It's an alternative to scp (part of ssh, see www.openssh.org (http://www.openssh.org)), but unlike scp, it don't require a login-account for the client on the server. The server-implementation of ssft is also lightweight and highly portable, currently supported under Linux, FreeBSD 4.3 and Windows NT (or better).

The program was initially written when I needed a secure way to transfer backup-files from Linux firewalls to Windows 2000 backup servers. The alternatives was ftp (which is a horrifying security-risk for critical systems), ssh (which would require a scp client with certificates, or a sshd implementation for Windows 2000, and even then compromise security as it require shell login permissions), samba (which is out of the question on firewalls) or nfs (which is also a security-risk, and not even available as a free client for Windows). Copying files in a secure manner from Linux to Windows 2000 was in fact not a trivial matter - and the solution was to invent SSFT.

SSFT runs from the command-line, or in the background as a native Unix daemon/Windows NT Service. The current implementation was written in a hurry, and is not scalable to large numbers of concurrent users. The protocol is however simple, and if there is a demand for it, I may implement the SSFT protocol in War FTP Daemon 3, which is scalable. The simplicity of the current version allow the same binary to act as a command-line interactive client, a command-line interactive server or as a hidden, native system daemon/service.

1.2. Platforms

The following platforms are supported:

- Windows NT, 2000 and successors
- Windows 98, ME and successors
- Linux
- FreeBSD
- Other Unix/Posix variants with the GNU C++ environment present (may need some porting efforts)

1.3. Licensing

The program is released under the Gnu Public License (GPL). The program and source code is in other words free. There is no fee, no user-registration, no royalties, no nothing.

1.4. Download

The latest version of SSFT can be downloaded from sourceforge.net/projects/ssft/ (http://sourceforge.net/projects/ssft/) The file section there contains both binary and source tarballs. The files can of course also be downloaded from ftp.jgaa.com (ftp://ftp.jgaa.com). If you run Debian Linux, you can try the experimental apt site at apt.jgaa.com (http://apt.jgaa.com).

1.5. Obtaning support and reporting bugs

SSFT is supported at support.jgaa.com (http://support.jgaa.com/). This site has a modern bug-reporting and bug-tracking facility, and you can also suggest features there, and subscribe to mailing lists. There is also a newsgroup: alt.comp.jgaa where you can ask for support.

If you have discovered a bug that can compromise the security in SSFT please contact <jgaa@jgaa.com> directly. If you don't get a reply within 12 hours, please resend the message.

1.6. Author

SSFT is written by Jarle (jgaa) Aase, best known for the original free FTP server for windows, War FTP Daemon. You can visit my homepage at www.jgaa.com (http://www.jgaa.com/).

Chapter 2. Compilation and installation

2.1. Source code or precompiled binary?

SSFT is distributed as source code, and as a compiled binary program for some platforms (like Windows). If you happen to use a platform which is supported by the binary distribution, this will be the most convenient way to install the package. If not, you are probably using some Unix/Posix compliant operating system, - and you need to compile the source code.

2.2. Compiling under Windows

Normally, you will use the binary Windows distribution. If you by some reason need to compile your own binary, you need Microsoft Visual C++ version 6, SP 3 (or better).

SSFT require the openssl library to compile. If you don't have this installed already, you must download it from www.openssl.org (http://www.openssl.org). Follow the instructions to compile the library (it's a bit tricky the first time, but it's well documented). When done, choose whether to use the dll version or static library version (I'm using the static library version to reduce the number of files in the binary distribution) and install it in Visual C++. (Open the Tools/Options menu, Select directory, and add the path to `"...OPENSSL-0.9.6B\INC32"` to Include files and `"...OPENSSL-0.9.6B\OUT32"` (or whatever suitable) to Library Files. Openssl will now work just as the built-in libraries in the Windows 32 SDK, and you don't have to manually add paths to it in your projects.

Download the latest source code from sourceforge.net/projects/ssft/ (https://sourceforge.net/projects/ssft/) and unpack it to a suitable location on your disk. Open the ssft project with Microsoft Visual C++. Compile either the Win32 Debug build or the Win32 Release build, depending on your requirements. If you compile the Win32 Debug build, you can trace the program 100% in the debugger.

2.3. Compiling under Linux/Unix/Posix

You need the openssl library installed on your system prior to compiling SSFT. If you don't have this installed already, you must download it from www.openssl.org (http://www.openssl.org). Follow the instructions to compile and install the library.

Download the latest source code from sourceforge.net/projects/ssft/ (https://sourceforge.net/projects/ssft/) and unpack it to a suitable location on your disk. Run **./configure** and **make** the program. You will probably need the GNU c++ environment and GNU make (that's what I've used anyway). Manually copy the binary to `/usr/local/bin` or do **make install**

Chapter 3. Configuration

3.1. Overview

SSFT secures the file transfers with encrypted TCP/IP connections using the SSL protocol. Both clients and servers must have a valid, public certificate, signed by a mutual trusted certificate authority. You can be your own certificate authority, so there is no need to purchase expensive certificates. The authentication scheme is flexible, and suited to handle small or large number of clients in a very convenient manner. If you are familiar with Windows, it may be a bit strange to work with configuration-files in plain text, but after a few minutes, you wil realize that it would be close to impossible to make something this simple, and yet flexible, using windows dialogs and wizards.

When a client connects to a server, it reads its own certificate and the certificate-authority's certificate. It passes it's own certificate to the server, and if the server approve the connection, it receives the serves public certificate and verifies the signature with the certificate-authority's certificate. If the server is legitim, it starts to send commands to the server. The server on its side, require clients to send their public certificate. When this is received, it first verifies the signature with the certificate-authority's public certificate. If the client is legitim, it compares the information about the client in its public certificate with rules in the servers .conf file. If a matching rule is found, the client is accepted, under the limitations defined by the first matching rule.

Why do SSFT mess with certificates, when most other protocols just ask for a user-name and a password? Well, the answer is very simple: security. By using certificates in both ends, it is very hard to expose a server to brute force password guessing attacks. The attacker would have to guess the entire certificate, including the signature - something that is a lot harder than just trying 100000 commonly used user names and passwords. The use of certificates, in combination with the rule-based user recognition, also makes it very easy to manage a huge number of users - as a simple rule can match anyone in a city, company or even an entire country.

The current implementation restricts anyone matching an authenticate-rule to one directory, and its subdirectories. This is done to reduce the complexity of the code, - the protocol itself have no such limitations. A future server-implementation may map any number of paths, with different access-rights, to a rule (or user certificate). "The one-certificate, one path" approach is sufficient to achieve the original goals of the program - to copy files in a secure manner between computers.

3.2. Creating certificates under Windows

The easy way to create certificates under Windows is to use ssft's built-in capabilities. This require the openssl.exe program to be in the same directory as ssft.exe or in the `PATH`.

1. Open a DOS window/Command prompt
2. Go to the directory where you installed ssft (**cd** `"C:\program files\ssft"`)
3. Create certificates
    a. **ssft** `--cert-dir certs --generate-ca`
    b. **ssft** `--cert-dir certs --new-server-cert`
    c. **ssft** `--cert-dir certs --new-client-cert`

You will be prompted for a password. This is the password for the root certificate you create, and must be remembered. If you forget this password, you can no longer create new certificates from the root certificate. (The certificate management is a simple wrapper around OpenSSL. If you are familiar with OpenSSL, you can use this directly).

The root certificate is used to sign the sever and client certificates, and is the mutual trusted authority that the security depends on. You can use your own and sign each client certificate - or you can use a commercial service like Thawte if you expect lots of clients, and don't have the means to do a proper validation of each of them (like when the clients are spread around all over the world).

The certificates you create are located in the directory `certs`, which can be any directory you like. The default names of these files are:

Table 3-1. Certificate file names

ssft_root.cert	Public certificate for the certificate authority. This must be distributed to all clients.
cakey.pem	Private certificate for the certificate authority. This should be protected well and secured with proper file-permissions (if possible). If someone steal this file, the entire security is compromised!
ssft_daemon.cert	Public certificate for the server. Must be signed by the certificate authority. Stored locally on the server-machine.
ssft_daemon.key	Private key for the server. Must be stored locally on the server-machine, and secured with proper file-permissions (if possible). Anyone that gets a copy of this file can impersonate the server(!)
ssft_client.cert	Public certificate for a client. Stored locally on the client-machine.
ssft_client.key	Private key for the client. Stored locally on the client-machine, and protected. Anyone that gets a copy of this fille, and the clients public key, can impersonate the client(!)

3.3. Creating certificates under Linux/Unix

Openssl comes with a perl-script, **CA.pl** that makes it pretty easy to create and manage certificates under Unix. The script has one weakness; it can not create certificates without a passphrase. In a secure server, you may not want the boot-process to stop just to enter a passphrase. I have made the following addition to the script:

```
} elsif (/^-newreq-nopwd$/) {
# create a certificate request
  system ("$REQ -new -keyout newreq.pem -out newreq.pem $DAYS -
nodes");
  $RET=$?;
  print "Request (and private key) is in newreq.pem\n";
```

If you use the **CA.pl** script for this purpose, you just have to rename the files you create to fit the purpose.

You can of course also use the procedure described above for Windows, to create certificates. Your clients can also create their own certificate requests, that you sign and send back to them. This is typically done when the clients are companies that require control over the .key files themselves.

3.4. The Configuration file

SSFT depends on a configuration file; a textfile with simple directives. If you are familiar with Windows, you can think of this file as a extended `.ini` file. If you are familiar with Unix, you know what a configuration file is ;)

The configuration file can be specified by a command-line argument when SSFT is started (**--conf-file=path**). If it is not specified, and SSFT is running as a client, SSFT will look for it as `...Application Data\ssft\ssft.conf` (Windows), or `~/.ssft.conf` (Unix). If this file is not found, or SSFT is running as a server, it will look for `...WINDOWS\ssft.conf` (Windows) or `/etc/ssft/ssft.conf` (Unix).

The configuration file consist of two sections; the general section, and the rule/user section.

3.4.1. General Section

Table 3-2. Options in the general section

cert-dir	The path to the directory containing the certificates.
log-file	The path to a log-file. If thee option is set, SSFT will maintain a log. If not, no log will be written (unless the --log-file command-line argument is given when SSFT starts up).
port	TCP/IP port-number to use. The default port is 230.
pid-file	The server can write its process ID to a file. This is common under UNIX systems, where signals to this process IP is used to shut down the server
server-cert	The name of the server-certificate
server-key	The name of the servers private key
client-cert	The name of the client-certificate
client-key	he name of the clients private key

3.4.2. Rule/User Section

The information is received from the client certificate, which means that the

combination of fields we use must be unique. The simplest setup would be to use the names only, and make sure to avoid duplicate names. We can use any field from the certificate 'subject'. Note that the compare method is case sensitive. Patterns are allowed.

Table 3-3. Options in the Rule/User section

alias	A name we use to identify the user uniquely (well, its for our convenience only so the server does not validate the value)
root-path	The users root-path. This location, with its subdirectories, is the only location on the filesystem that this client is allowed to access.
perms	The operations we allow for this client Possible values are: read, write, list, delete, createdir

3.4.3. Sample configuration file

```
# SSFT demo server configuration file
#
# This is just an example of a configuration
# file for ssft under Microsoft Windows.
#


# ================================================
# ================================================
# General section
# ================================================
```

```
# ================================================

[General]

cert-dir = C:\devel\current\ssft\certs
log-file = C:\devel\current\ssft\svr_test\server.log

# Uncomment and set port number if you want to
# use an alternative port (230 is the default port)
# port =

# The server can write its process ID to a file.
# This is common under UNIX systems, where signals
# to this process IP is used to shut down the server.
# pid-file =


# SERVER options
# server-cert =
# server-key =

# Uncomment if you want to bind to a spesific IP
# number. The default is to use all IP numbers
# assigned to the machine.
# host =


# CLIENT options
# client-cert =
# client-key =

# ================================================
# ================================================
# User section
# ================================================
# ================================================
```

```
# This is relevant for servers only
#
# The information is received from the client sertificate,
# which means that the combination of fields we use must
# be unique. The simplest setup would be to use the
# names only, and make sure to avoid duplicate names.
# We can use any field from the certificate 'subject'.
# Note that the compare method is case sensitive.
#
# Patterns are allowed.


# Spesific user
[User: O=Jgaas Inter-
net, CN=jgaa client 1, Email=jgaa@client1.jgaa.com]

# A name we use to identify the user uniquely
# (well, its for our convenience only so the
# server does not validate the value)
alias  = Jarle

# The users root-path
root-path  = C:\ssft_test\users\jgaa_client_1

# The operations we allow for this client
# Possible values are: read, write, list, delete, createdir
perms  = read, write, list, delete, createdir


# All other clients from Jgaa's Internet
[User: O=Jgaas Internet, CN=jgaa client*]
alias  = jgaa's Internet
root-path  = C:\ssft_test\users\Jgaas Internet
perms  = list

# Default rights for US users
[User: C=US]
```

```
alias  = US citizens
root-path  = C:\ssft_test\users\us\unknown
perms  = list


# Default rights if not specified above
[User: CN=*]
alias  = nobody
root-path  = C:\ssft_test\users\unknown
perms  = list, read
```

> **Note:** The lines may be wrapped in this manual. In the configuration-file, the "token = value" sequence must be on one single line. This also applies for the "[User: ...]" sequence.

Chapter 4. Testing SSFT

4.1. Server test

I assume that you have created the certificates and edited the configuration-file. Below is the configuration file I used in this sample:

```
[General]

# Default rights if not specified above
[User: CN=*]
alias  = nobody
root-path  = C:\Program Files\ssft\public
perms  = read, write, list, delete, createdir
```

Now, open a MS-DOS/Command window, or terminal window if you are running X, and start SSFT in server-mode.

```
C:\Program Files\ssft>ssft --cert-dir certs --server --verbose
ssft version 0.10, Copyright (C) 2001 Jarle (jgaa) Aase
ssft comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions;
ssft --license for details.
Listening for connections at 0.0.0.0:230
Ready, waiting for client connection...
```

If you get a output similar to this, things are looking good ;) The server is willingly running and waitig for clients. Keep it running while we test the client-mode.

4.2. Client test

Open a new MS-DOS/Command window or terminal window, and try a file-copy command, using SSFT

```
C:\Program Files\ssft>ssft --verbose --cert-
dir certs ReadMe.txt localhost:/
ssft version 0.10, Copyright (C) 2001 Jarle (jgaa) Aase
ssft comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions;
ssft --license for details.
Looking up hostname localhost ... 127.0.0.1
Connecting to 127.0.0.1:230
SSL connection using DES-CBC3-SHA
Server certifi-
cate: /C=NO/ST=Hordaland/L=Bergen/O=Jgaas Internet...
Certificate is-
suer: /C=NO/ST=Hordaland/L=Bergen/O=Jgaas Internet...
connected to server
Sending file "ReadMe.txt" --> "localhost:/"
File transfered OK
Successfully done.
```

If you switch back to the server console, you should see something like this:

```
Client connected from 127.0.0.1:1038
Client sent Transfer from Client to Server command
Receiving file from client no-
body (127.0.0.1:1038): name="C:\Program Files\ssft\public\ReadMe.txt" size=2
Received file client (127.0.0.1:1038): name="ReadMe.txt" size=262 bytes. OK
Client sent Quit command
```

When you have come this far, the program is working, and you can start to set up and test your production environment.

Note: It's a good idea to test everything in interactive mode before you run SSFT as a daemon, or the client as an automated job.

Chapter 5. Deployment

5.1. Windows NT Service

SSFT can run as a native Windows NT service. This means that the program starts when the machine boots, and runs in the background, unaffected by user sessions on the machine. The only way SSFT can communicate with the operator while running as a service, is trough the logfile. This should therefore be assigned in the configuration file at this time.

```
[General]

log-file = C:\Program Files\ssft\server.log
```

The following command will install SSFT as a service:

```
C:\Program Files\ssft>ssft --server --daemon --verbose --install
Successfully installed as a NT service as "ssft".
You must now enter the service applet in the con-
trol panel and configure the startup mode.

C:\Program Files\ssft>net start ssft
The ssft service is starting.
The ssft service was started successfully.

C:\Program Files\ssft>type server.log
==================================================
==================================================
2001-10-19 07:48 Now running as a native NT service.
2001-10-
19 07:48 ssft version 0.10, Copyright (C) 2001 Jarle (jgaa) Aase
ssft comes with ABSOLUTELY NO WARRANTY
This is free software, and you are welcome to redistribute it
under certain conditions;
```

```
ssft.exe --license for details.
2001-10-19 07:48 Ready, waiting for client connection...

C:\Program Files\ssft>
```

The three commands, **ssft, net start and type** installs SSFT as a service, verifies that it can start, and prints the log to the screen, so that you can spot any error-messages. If everything seems ok, you can open the Services apples in the control panel, and configure the service to start automatically.

If you ever need to uninstall SSFT as a NT service, just issue the command **ssft** `--uninstall`.

5.2. Unix daemon

SSFT can run as a native Unix daemon. This means that the program starts when the machine boots, and runs in the background, unaffected by user sessions on the machine. The only way SSFT can communicate with the operator while running as a service, is trough the logfile. This should therefore be assigned in the configuration file at this time.

There is no uniform way to install a Unix daemon. The procedure varies from operating system to operating system, and even from distribution to distribution. SSFT must therefore be manually configured according to your system documentation.

In short, the command **ssft --server --daemon** will start ssft in server-mode, and switch to daemon mode. This command can be run from the initialization script for your Unix brand (if the certificates and configuration file are ok).

Appendix A. Command Line Syntax and Options

A.1. Overview

SSFT understand Posix-style command-line options. In environments with a complete Gnu c library (line Linux), the Posix compatibility should be pretty good - but under Windows and other systems that don't support the getopt_long() library function, SSFT use a simple substitute I wrote some time ago. This substitute understand the `--option` and `--` flags, but will not reorganize options. In English, this means that options must be given before other arguments.

SSFT works much like scp (or cp/copy). You specify a file (or several files), and a destination (file or folder). Either the source or destination must contain a hostname or IP number that prefix the source or destination file. Some examples:

Example A-1. Copying files (Windows)

```
ssft "C:\Program Files\ssft\ReadMe.txt" 192.168.0.10:/testdir

ssft 192.168.0.10:/testdir/ReadMe.txt "C:\Program Files\ssft\"

ssft 192.168.0.10:/testdir C:\temp
```

Example A-2. Copying files (Unix)

```
ssft /usr/local/doc/ssft/ReadMe.txt 192.168.0.10:/testdir

ssft 192.168.0.10:/testdir/ReadMe.txt /home/jgaa

ssft 192.168.0.10:/testdir /tmp
```

Note: Note that the remote path always use normal slashes, while the local paths use the semantics of your operating system.

A.2. Getting help

Use **ssft** `--help` to get a brief list and explanation of all the supported command-line options.

```
C:\>ssft --help
Copy files (client mode)
 ssft [options] host:source target
 ssft [options] source [source...] host:target
Server mode:
 ssft [options] [hostname]
Utility mode (install, uninstall, certificate management):
 ssft option [...]
Options:
 -c  --conf-file=name   Specifies the configuration file to use
 -p  --port=#           Set port number. Default is 230
 -v  --version          Show version number and quit
     --verbose          Turn on verbose mode
     --debug            Turn on debug messages
     --server           Server mode (default is client mode)
     --daemon           Runs as a daemon
 -l  --log-file=name    Name of logfile
 -h  --help             Show this help
     --cert-dir=path    Certificate directory
     --root-cert=name   Root certificate file
     --client-cert=name Client certificate
     --client-key=name  Client private key file
     --server-cert=name Server certificate
```

```
     --server-key=name  Server private key file
     --pid-file=name    Write process ID to file
 -s  --service-name=name Sets the service name. Used with --
install
 -i  --install          Install as a NT service (and quit)
 -u  --uninstall        Uninstall as a NT service (and quit)
 -L  --license          Show licensing terms (GPL)

Certificate management
     --generate-ca      Generates a root certificate
     --new-server-cert  Generates a server certificate
     --new-client-cert  Generates a client certificate
```

The actual options that are available may vary from version to version, and from operating system to operating system.

A.3. The options

Table A-1. Command line options

--conf-file	Specifies the configuration-file to use. This option is persistent when used under Windows with the --install --daemon --server options. You can also specify what configuration-file the service shall use, and if use different ports and service-names, you can have several service instances running at the same Windows machine. Under Unix, the startup scripts are sane, so this special feature is not required.

--port	Specifies a TCP/IP port to use. The server will listen to this port. The client will connect to it.
--version	Prints the program name and version number to the console and.
--verbose	Prints more verbose information to the console.
--debug	Prints lots of information to the console. Used to debug the internals of the program.
--server	Starts up in server-mode. the default is client-mode. In server-mode, SSFT waits for incomming connections from clients.
--daemon	Runs as a Unix daemon (hidden and unaffected by the user). Under NT, this flag is used with --install to flag that the program shall be installed as a system service.
--log-file	Specifies a log-file. If not set, no log will be written, (unless the log-file directive is specified in the configuration file).
--help	Prints a short help-text to the console.
--cert-dir	Certificate directory
--root-cert	Root certificate file
--client-cert	Client certificate
--client-key	Client private key file
--server-cert	Server certificate
--server-key	Server private key file

--service-name	Used with --install to specify the name the NT server will register as. The default is "ssft". Other names can be used to test experimental versions, or to run several instances of the service.
--install	Installs SSFT as a Windows NT service and exits. This option is only available under Windows NT and successors.
--uninstall	Uninstalls the SSFT NT service. If you specified another name than "ssft" with --service-name, you must specify the same option here.
--license	Show licensing terms (Gnu Public License)
--generate-ca	Generates a new certificate authority certificate and key pair. This is required if you want to be your own certificate authority. You will be prompted for a few questions and a password. It is important that you remember this password!
--new-server-cert	Generates a new server certificate, and signs it with the certificate authority certificate.
--new-client-cert	Generates a new client certificate, and signs it with the certificate authority certificate.

Appendix B. Firewalls

When it comes to legitim network traffic, firewalls are hell. Many new protocols today encapsulate themselves into the http (www) protocol to bypass firewalls. This may seem like a good idea, initially - as one avoid some problems with troublesome firewall administrators. But it also undermines the security, since the same firewall administrators loose the ability to decide what to let trough to the internal network. SSFT makes no attempts to masquerade itself as something else. The protocol is however designed to make as little problems as possible when it is allowed trough a firewall. It makes use of a single TCP/IP address, and have no need for protocol-level NAT if it is routed to a private network (like i.e. FTP).

In order to allow SSFT trough a firewall, open TCP port 230 from the client(s) to the server(s). This is trivial on most firewalls, including ipchains and iptable based Linux firewalls. If you have a firewall that require a protocol-layer driver, you can use any driver that pass the TCP/IP stream untouched trough. A Telnet driver cannot be trusted as it may react on telnet escape sequences.

Appendix C. The SSFT protocol

C.1. Overview

The SSFT protocol is a request/response driven protocol, like http and nntp. Unlike these, the headers are sent as raw 8-bit binary data. Integers are in "network byte order" (big endian).

More details to follow...