

Safepipe interoperability with FreeS/WAN on a Linux host

A tech note prepared by Eicon Networks August 2001

Summary

Safepipe is interoperable with FreeS/WAN on a Linux host. In a test scenario the following was established:

- *Tunnels can successfully be established between the two devices.*
- *Use of authentication mode (shared secret) can successfully be negotiated.*
- *Use of authentication algorithm (MD5) can successfully be negotiated.*
- *Use of encryption algorithm (triple-DES) can successfully be negotiated.*
- *ISAKMP SA lifetime and IPSec key lifetime can successfully be negotiated.*
- *Both devices are able to initiate a tunnel.*
- *Data packets can successfully be exchanged through the tunnels.*

Structure of this document:

Introduction	4
Aims and objectives	4
Test scope and example network	5
Known limitations	5
Configuration of FreeS/WAN on Linux host	6
Basic configuration options in two files	6
/etc/ipsec.conf	7
/etc/ipsec.secrets	9
Starting the tunnel from Linux	9
Configuration on Safepipe	10
Configuring the tunnel	10
Verifying that traffic is able to pass through the tunnel	12
Using FreeS/WAN's debug features	12
Viewing the Safepipe tunnel log	13
Conclusion	14

Introduction

As VPN standards evolve, not all implementations by the various vendors of VPN devices may be interoperable. One of Safepipe's strengths is its ability to interoperate with several VPN solutions offered by other vendors/organisations. This document describes a successful test of Safepipe's interoperability with FreeS/WAN running on a Linux host, and details the steps needed to configure the two devices for interoperability.

Please note that this tech note assumes a familiarity with Safepipe, Linux and FreeS/WAN. It describes VPN tunnelling on a test network; if recreating the scenario, substitute any names, IP addresses, etc. with names and addresses relevant to your own network.

Aims and objectives

The aim of the test described in this document was to determine whether Safepipe and FreeS/WAN on a Linux host were interoperable, i.e. whether a functional tunnel could be established between the two parties.

In more detail, we needed:

1. To determine whether an authentication mode (shared secret) could be negotiated between the two devices.
2. To determine whether an authentication algorithm could be negotiated between the two devices.
3. To determine whether encryption with a triple-DES algorithm could be negotiated between the two devices.
4. To determine whether key lifetime could be negotiated between the two devices.
5. To determine whether both parties were able to act as tunnel initiators.
6. To determine whether exchange of data through the tunnel was possible.

Test scope and example network

The test was performed using a Safepipe 50 with software version 2.3 and a Linux host with FreeS/WAN 1.9 on SuSE Linux 7.2, kernel 2.2.19¹. The Linux host had two network interface cards. One NIC was used as a public interface; and one was used as a private interface.

The test network was set up with a workstation on each of the private networks connected to the private interfaces of the Safepipe and Linux host respectively. Before the test, the various interfaces were configured with IP addresses.

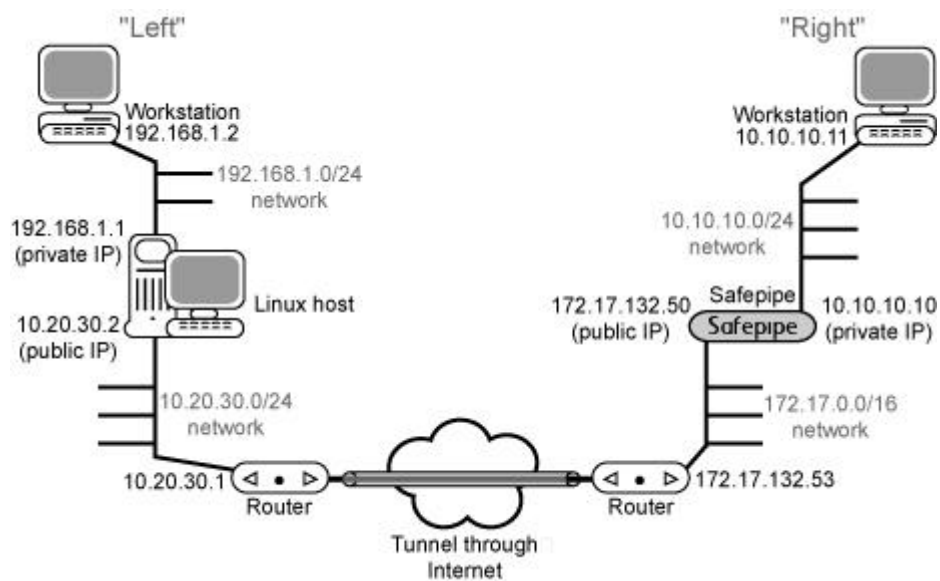


Diagram of test network

Known limitations

None, although no extensive performance test was carried out.

¹ Please note that there appears to be a problem with FreeS/WAN integration in the 2.4 kernel of SuSE 7.2.

Configuration of FreeS/WAN on Linux host

The following configuration requires that FreeS/WAN has been installed on the Linux host. FreeS/WAN allows Linux to support IKE and IPSec. FreeS/WAN may be pre-compiled with your Linux distribution, or you may have to install and compile it yourself. For FreeS/WAN and FreeS/WAN information and documentation, please refer to <http://www.freeswan.org>.

The following configuration furthermore requires that the Linux host running FreeS/WAN has been initially configured with IP addresses, etc., and that it has been connected according to the example network diagram in this document.

In our example, the Linux host acts as a security gateway. This requires that forwarding is enabled. Exactly how to enable forwarding depends on your Linux distribution, but typically one would use

```
# echo "1" >/proc/sys/net/ipv4/ip_forward
```

After a test phase, one would typically enable forwarding from an init script.

Forwarding is only necessary if the Linux host acts as a gateway for a network located behind it, not if the Linux host simply acts as a “VPN client”.

Basic configuration options in two files

Two files contain the basic configuration options needed for FreeS/WAN:

- ***/etc/ipsec.conf*** is FreeS/WAN’s IPSec configuration file.
- ***/etc/ipsec.secrets*** contains the shared secrets for authentication between pluto (an implementation of IKE, running as a daemon on the Linux host) and another IKE daemon, e.g. another instance of pluto. In our case this simply means the shared secret used for authentication between the two parties in the VPN.

We shall look at the two files the two files, starting with */etc/ipsec.conf*:

/etc/ipsec.conf

This file is FreeS/WAN's IPsec configuration file. Two sections in the `/etc/ipsec.conf` file need to be defined: `config setup` and `conn [connection name]`.

The `config setup` section is the part of the `/etc/ipsec.conf` file that allows attachment of virtual IPsec interfaces to network interfaces. It has several parameters:

The `interfaces` parameter defines interfaces for IPsec to use. In our example `interfaces=%defaultroute` is used. The `%defaultroute` value specifies use of the interface that the default route points to. This value is acceptable for most simple cases.

You may find the `klipsdebug` (KLIPS stands for Kernel IPsec Support) and `plutodebug` parameters useful if you require debugging output. If so, change their values from `none` to `all`. More information is available on page 12 of this document.

`plutoload` and `plutostart` control which connections to load and attempt to negotiate at startup, and can thus be used for automatic start of VPN connections when FreeS/WAN is started. The `%search` value used in our example indicates that any connection featuring `auto=start` will be started.

`uniqueids=yes` makes sure that old connections are closed down when a new one using the same ID appears.

More information is available through `man ipsec.conf`.

The `config setup` part of our `/etc/ipsec.conf` file thus looked like this:

```
# basic configuration
config setup
    interfaces=%defaultroute
    klipsdebug=none
    plutodebug=none
    plutoload=%search
    plutostart=%search
    uniqueids=yes
```

The next part of the `/etc/ipsec.conf` file is the `conn [connection name]` section. This is the part that defines the VPN connection, including routes. In the below example, we used the connection name “eiconsafepipe”.

The `type` parameter controls the connection type. By specifying `type=tunnel`, we required that tunnel mode (as opposed to transport mode) was used.

When defining the VPN, FreeS/WAN distinguishes between a “left” side and a “right” side. In our example, the left side denotes the side that the Linux host is situated on, and the right side denotes the side that the Safepipe is situated on.

Therefore,

- `left` represents the public interface of the Linux host.
- `leftsubnet` represents the private net behind the Linux host.
- `leftnexthop` represents the gateway that the left (Linux host) side of the VPN will use to reach the right (Safepipe) side of the VPN.

The right side of the VPN is defined in a similar fashion.

`keyexchange=ike` specifies that IKE should be used when exchanging keys.

Note that had we included a line specifying `auto=start`, the `plutoload=%search` and `plutostart=%search` parameters used in the `config setup` section above would have triggered autostart of the VPN connection when starting FreeS/WAN.

The `conn [connection name]` part of our `/etc/ipsec.conf` file thus looked like this:

```
# connection
conn eiconsafepipe
    type=tunnel
    left=10.20.30.2
    leftsubnet=192.168.1.0/24
    leftnexthop=10.20.30.1
    right=172.17.132.50
    rightsubnet=10.10.10.0/24
    rightnexthop=172.17.132.53
    keyexchange=ike
```

/etc/ipsec.secrets

This file allows specification of shared secrets that should be used between the two parties in the VPN. The secrets must match, so identical secrets must be configured on the Safepipe as well as on FreeS/WAN.

In our example we chose the secret "topsecret". Be sure to use a less obvious secret if creating a VPN of your own.

The below excerpt from the file lists (from left to right):

- The IP address of the public interface of the Linux host.
- The IP address of the public interface of the Safepipe.
- PSK, indicating Pre-Shared Key (a.k.a. shared secret).
- The secret, which **MUST** be enclosed in quotes, even if the secret does not contain spaces.

Our `/etc/ipsec.secrets` configuration thus looked like this:

```
10.20.30.2 172.17.132.50 : PSK "topsecret"
```

Starting the tunnel from Linux

You would bring up the tunnel from Linux by using:

```
# start ipsec:
# either an init script (e.g. /etc/init.d/ipsec start)
# or
ipsec setup start

# defining the tunnel for freeswan (if one does not have auto=add or
auto=start for the connection in the ipsec.conf file):
ipsec auto --add [connection name]

# start tunnel (if one does not have auto=start in ipsec.conf)
ipsec auto --up [connection name]
```

Configuration on Safepipe

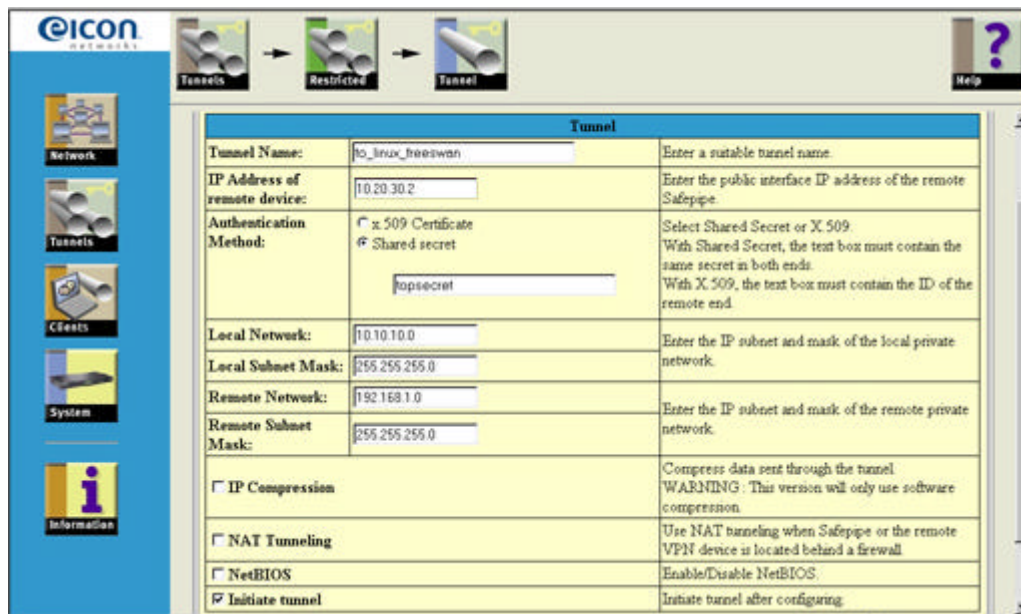
The following configuration requires that the Safepipe device has been initially configured with IP addresses, etc., that it has been connected according to the example network diagram in this document, and that the user has launched and logged on to Safepipe's browser-based user interface.

Configuring the tunnel

From Safepipe's menu, select **Tunnels** -> **Restricted**. You cannot use Safepipe's Automatic Tunnel option with FreeS/WAN.

Click the **Add Tunnel** button. Click the **<Not Configured>** link that appears.

A tunnel configuration page appears:



Tunnel		
Tunnel Name:	<input type="text" value="to_linux_freeswan"/>	Enter a suitable tunnel name.
IP Address of remote device:	<input type="text" value="10.20.30.2"/>	Enter the public interface IP address of the remote Safepipe.
Authentication Method:	<input type="checkbox"/> X.509 Certificate <input checked="" type="checkbox"/> Shared secret <input type="text" value="topsecret"/>	Select Shared Secret or X.509. With Shared Secret, the text box must contain the same secret in both ends. With X.509, the text box must contain the ID of the remote end.
Local Network:	<input type="text" value="10.10.10.0"/>	Enter the IP subnet and mask of the local private network.
Local Subnet Mask:	<input type="text" value="255.255.255.0"/>	
Remote Network:	<input type="text" value="192.168.1.0"/>	Enter the IP subnet and mask of the remote private network.
Remote Subnet Mask:	<input type="text" value="255.255.255.0"/>	
<input type="checkbox"/> IP Compression		Compress data sent through the tunnel. WARNING: This version will only use software compression.
<input type="checkbox"/> NAT Tunneling		Use NAT tunneling when Safepipe or the remote VPN device is located behind a firewall.
<input type="checkbox"/> NetBIOS		Enable/Disable NetBIOS.
<input checked="" type="checkbox"/> Initiate tunnel		Initiate tunnel after configuring.

Tunnel configuration page on Safepipe

- Enter a suitable name for the tunnel in the **Tunnel Name** field (in this example the name "to_linux_freeswan" was used).
- Enter the IP address of the public interface on the Linux host in the **IP Address of remote device** field.

- Select **Shared secret** as the **Authentication Method** to use, and enter the agreed shared secret in the associated field. Be sure to enter exactly the same secret as was specified in the `/etc/ipsec.secrets` file on FreeS/WAN.
- Enter the IP address of the network on the private side of the Safepipe in the **Local Network** field.
- Enter the associated subnet mask in the **Local Subnet Mask** field.
- Enter the IP address of the network on the private side of the Linux host in the **Remote Network** field.
- Enter the associated subnet mask in the **Remote Subnet Mask** field.
- Leave the **IP Compression** box **unchecked**².
- Leave the **NAT Tunnelling** box **unchecked**.
- Leave the **NetBIOS** box **unchecked**.
- Check the **Initiate tunnel** box in order for Safepipe to be the initiating party in the tunnel. With the **Initiate tunnel** box checked, Safepipe was able to successfully initiate a tunnel to the Linux Host. FreeS/WAN was also able to successfully initiate a tunnel to Safepipe, when the **Initiate tunnel** box on Safepipe was not checked.

Click the **Apply Changes** button, and make sure that the **Enabled** box for the tunnel in question is checked.

² IP compression is possible, and a tunnel with compression was tested successfully. If using compression, however, it must be enabled on Safepipe as well as on FreeS/WAN. On Safepipe, compression is enabled by checking the IP Compression box. On FreeS/WAN, a line specifying `compress=yes` should be added to the `/etc/ipsec.conf` file.

Verifying that traffic is able to pass through the tunnel

Once each of the tunnel endpoint devices had been configured as described, we established that data could successfully be exchanged through the tunnels. This was done by having the workstation located on the private side of the Safepipe successfully ping the workstation located on the private side of the Linux host through the tunnel, and vice versa.

Please note that the Linux host itself is not part of the VPN. Thus, you cannot communicate from the Linux host itself to the VPN behind the Safepipe, e.g. by ping. If you want the Linux host to be the only point on the “left” side of the connection (in effect being a “VPN client”) you should make the following configuration changes:

- In the `/etc/ipsec.conf` file on the Linux host,
 - change the `leftsubnet` value to `[public IP address of Linux host]/32`
- On the Safepipe,
 - change the **Remote subnet** field to `[public IP address of Linux host]`
 - change the **Remote subnet mask** field to `255.255.255.255`.

We used the log/debug features of the two devices to verify that tunnels had indeed been established according to configuration and that the ping data traffic we sent was thus able to pass securely through the tunnels.

Using FreeS/WAN's debug features

You can use the `klipsdebug` (KLIPS stands for Kernel IPsec Support) and `plutodebug` parameters in the `/etc/ipsec.conf` file if you require debugging output in order to verify that the tunnel has been correctly established. Most systems are set up to log these messages to `/var/log/messages`.

If you want debugging output, change the values of the two parameters from `none` to `all` as in the following example:

```
klipsdebug=all
plutodebug=all
```

Viewing the Safepipe tunnel log

On Safepipe's menu, select **Tunnels** -> **Restricted**.

Click the **Name** link of the tunnel in question.

Click the **Log** tab to view the log for the tunnel in question.

The log is updated every time the tab is accessed. In case you wish to update the log while viewing it, click the Tunnel icon above the log.

Conclusion

Safepipe is interoperable with FreeS/WAN on a Linux host.

We were able to successfully establish a VPN tunnel between the Safepipe and the Linux host running FreeS/WAN. Both parties were able to initiate a tunnel. By monitoring logs on the two devices we were able to verify that negotiation had been successful, i.e. according to our configuration.

Successful negotiation included: negotiation of authentication mode (shared secret), negotiation of authentication algorithm (MD5, proposed and accepted by FreeS/WAN as well as Safepipe), negotiation of encryption algorithm (triple-DES, the default encryption algorithm of FreeS/WAN as well as Safepipe), and negotiation of key lifetime.

By having workstations located on the private networks at each end of the tunnel successfully ping each other through the tunnel, we established that data could be exchanged through the tunnels.

We did not test performance further.