Contents Previous Next

FreeS/WAN and firewalls

FreeS/WAN, or other IPsec implementations, frequently run on gateway machines, the same machines running firewall or packet filtering code. This document discusses the relation between the two.

The firewall code in 2.4 and later kernels is called Netfilter. The user-space utility to manage a firewall is iptables(8). See the netfilter/iptables web site for details.

Filtering rules for IPsec packets

The basic constraint is that **an IPsec gateway must have packet filters that allow IPsec packets**, at least when talking to other IPsec gateways:

- UDP port 500 for IKE negotiations
- protocol 50 for ESP encryption and/or authentication

Your gateway and the other IPsec gateways it communicates with must be able to exchange these packets for IPsec to work. Firewall rules must allow UDP 500 and ESP on the interface that communicates with the other gateway.

There are two ways to set this up:

easier but less flexible
: Just set up your firewall scripts at boot time to allow IPsec packets to and from your gateway. Let FreeS/WAN reject any bogus packets.

more work, giving you more precise control
: Have the ipsec_pluto(8) daemon call scripts to adjust firewall rules dynamically as required. This is done by naming the scripts in the ipsec.conf(5) variables *prepluto=*, *postpluto=* , *leftupdown=* and *rightupdown=*.

Both methods are described in more detail below.

Firewall configuration at boot

It is possible to set up both firewalling and IPsec with appropriate scripts at boot and then not use *leftupdown=* and *rightupdown=*, or use them only for simple up and down operations.

Basically, the technique is

- allow IPsec packets (typically, IKE on UDP port 500 plus ESP, protocol 50)
  - incoming, if the destination address is your gateway (and optionally, only from known senders)
  - outgoing, with the from address of your gateway (and optionally, only to known receivers)
- let Pluto deal with IKE
- let KLIPS deal with ESP

Since Pluto authenticates its partners during the negotiation, and KLIPS drops packets for which no

tunnel has been negotiated, this may be all you need.

A simple set of rules

In simple cases, you need only a few rules, as in this example:

```
# allow IPsec
#
# IKE negotiations
iptables -I INPUT  -p udp --sport 500 --dport 500 -j ACCEPT
iptables -I OUTPUT -p udp --sport 500 --dport 500 -j ACCEPT
# ESP encryption and authentication
iptables -I INPUT  -p 50 -j ACCEPT
iptables -I OUTPUT -p 50 -j ACCEPT
```

This should be all you need to allow IPsec through *lokkit*, which ships with Red Hat 9, on its medium security setting. Once you've tweaked to your satisfaction, save your active rule set with:

```
service iptables save
```

Other rules

You can add additional rules, or modify existing ones, to work with IPsec and with your network and policies. We give a some examples in this section.

However, while it is certainly possible to create an elaborate set of rules yourself (please let us know via the mailing list if you do), it may be both easier and more secure to use a set which has already been published and tested.

The published rule sets we know of are described in the next section.

Adding additional rules

If necessary, you can add additional rules to:

reject IPsec packets that are not to or from known gateways
: This possibility is discussed in more detail later

allow systems behind your gateway to build IPsec tunnels that pass through the gateway
: This possibility is discussed in more detail later

filter incoming packets emerging from KLIPS.
: Firewall rules can recognise packets emerging from IPsec. They are marked as arriving on an interface such as *ipsec0*, rather than *eth0*, *ppp0* or whatever.

It is therefore reasonably straightforward to filter these packets in whatever way suits your situation.

Modifying existing rules

In some cases rules that work fine before you add IPsec may require modification to work with IPsec.

This is especially likely for rules that deal with interfaces on the Internet side of your system. IPsec adds a new interface; often the rules must change to take account of that.

For example, consider the rules given in this section of the Netfilter documentation:

```
Most people just have a single PPP connection to the Internet, and don't
want anyone coming back into their network, or the firewall:

    ## Insert connection-tracking modules (not needed if built into kernel).
    # insmod ip_conntrack
    # insmod ip_conntrack_ftp

    ## Create chain which blocks new connections, except if coming from inside.
    # iptables -N block
    # iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
    # iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
    # iptables -A block -j DROP

    ## Jump to that chain from INPUT and FORWARD chains.
    # iptables -A INPUT -j block
    # iptables -A FORWARD -j block
```

On an IPsec gateway, those rules may need to be modified. The above allows new connections from *anywhere except ppp0*. That means new connections from ipsec0 are allowed.

Do you want to allow anyone who can establish an IPsec connection to your gateway to initiate TCP connections to any service on your network? Almost certainly not if you are using opportunistic encryption. Quite possibly not even if you have only explicitly configured connections.

To disallow incoming connections from ipsec0, change the middle section above to:

```
    ## Create chain which blocks new connections, except if coming from inside.
    # iptables -N block
    # iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
    # iptables -A block -m state --state NEW -i ppp+ -j DROP
    # iptables -A block -m state --state NEW -i ipsec+ -j DROP
    # iptables -A block -m state --state NEW -i -j ACCEPT
    # iptables -A block -j DROP
```

The original rules accepted NEW connections from anywhere except ppp0. This version drops NEW connections from any PPP interface (ppp+) and from any ipsec interface (ipsec+), then accepts the survivors.

Of course, these are only examples. You will need to adapt them to your own situation.

Published rule sets

Several sets of firewall rules that work with FreeS/WAN are available.

Scripts based on Ranch's work

One user, Rob Hutton, posted his boot time scripts to the mailing list, and we included them in previous versions of this documentation. They are still available from our web site. However, they were for an earlier FreeS/WAN version so we no longer recommend them. Also, they had some bugs. See this message.

Those scripts were based on David Ranch's scripts for his "Trinity OS" for setting up a secure Linux. Check his home page for the latest version and for information on his book on securing Linux. If you are going to base your firewalling on Ranch's scripts, we recommend using his latest version, and sending him any IPsec modifications you make for incorporation into later versions.

The Seattle firewall

We have had several mailing lists reports of good results using FreeS/WAN with Seawall (the Seattle Firewall). See that project's home page on Sourceforge.

The RCF scripts

Another set of firewall scripts with IPsec support are the RCF or rc.firewall scripts. See their home page.

Asgard scripts

Asgard's Realm has set of firewall scripts with FreeS/WAN support, for 2.4 kernels and iptables.

User scripts from the mailing list

One user gave considerable detail on his scripts, including supporting IPX through the tunnel. His message was too long to conveniently be quoted here, so I've put it in a separate file.

Calling firewall scripts, named in ipsec.conf(5)

The ipsec.conf(5) configuration file has three pairs of parameters used to specify an interface between FreeS/WAN and firewalling code.

Note that using these is not required if you have a static firewall setup. In that case, you just set your firewall up at boot time (in a way that permits the IPsec connections you want) and do not change it thereafter. Omit all the FreeS/WAN firewall parameters and FreeS/WAN will not attempt to adjust firewall rules at all. See above for some information on appropriate scripts.

However, if you want your firewall rules to change when IPsec connections change, then you need to use these parameters.

Scripts called at IPsec start and stop

One pair of parmeters are set in the *config setup* section of the ipsec.conf(5) file and affect all connections:

prepluto=
: script to be called before pluto(8) IKE daemon is started.

postpluto=
: script to be called after pluto(8) IKE daemon is stopped.

These parameters allow you to change firewall parameters whenever IPsec is started or stopped.

They can also be used in other ways. For example, you might have *prepluto* add a module to your kernel for the secure network interface or make a dialup connection, and then have *postpluto* remove the module or take the connection down.

Scripts called at connection up and down

The other parameters are set in connection descriptions. They can be set in individual connection descriptions, and could even call different scripts for each connection for maximum flexibility. In most applications, however, it makes sense to use only one script and to call it from *conn %default* section so that it applies to all connections.

You can:

either
set *leftfirewall=yes* or *rightfirewall=yes* to use our supplied default script
or
assign a name in a *leftupdown=* or *rightupdown=* line to use your own script

Note that **only one of these should be used**. You cannot sensibly use both. Since **our default script is obsolete** (designed for firewalls using *ipfwadm(8)* on 2.0 kernels), most users who need this service will **need to write a custom script**.

The default script

We supply a default script named *_updown*.

leftfirewall=
rightfirewall=
indicates that the gateway is doing firewalling and that pluto(8) should poke holes in the firewall as required.

Set these to *yes* and Pluto will call our default script *_updown* with appropriate arguments whenever it:

- starts or stops IPsec services
- brings a connection up or down

The supplied default *_updown* script is appropriate for simple cases using the *ipfwadm(8)* firewalling package.

User-written scripts

You can also write your own script and have Pluto call it. Just put the script's name in one of these ipsec.conf(5) lines:

leftupdown=
rightupdown=
specifies a script to call instead of our default script *_updown*.

Your script should take the same arguments and use the same environment variables as *_updown*. See the "updown command" section of the ipsec_pluto(8) man page for details.

Note that **you should not modify our _updown script in place** . If you did that, then upgraded FreeS/WAN, the upgrade would install a new default script, overwriting your changes.

Scripts for ipchains or iptables

Our *_updown* is for firewalls using *ipfwadm(8)*, the firewall code for the 2.0 series of Linux kernels. If you are using the more recent packages *ipchains(8)* (for 2.2 kernels) or *iptables(8)* (2.4 kernels), then you must do one of:

- use static firewall rules which are set up at boot time as described above and do not need to be changed by Pluto
- limit yourself to ipchains(8)'s ipfwadm(8) emulation mode in order to use our script

- write your own script and call it with *leftupdown* and *rightupdown*.

You can write a script to do whatever you need with firewalling. Specify its name in a *[left|right] updown=* parameter in ipsec.conf(5) and Pluto will automatically call it for you.

The arguments Pluto passes such a script are the same ones it passes to our default _updown script, so the best way to build yours is to copy ours and modify the copy.

Note, however, that **you should not modify our _updown script in place**. If you did that, then upgraded FreeS/WAN, the upgrade would install a new default script, overwriting your changes.

A complication: IPsec vs. NAT

Network Address Translation, also known as IP masquerading, is a method of allocating IP addresses dynamically, typically in circumstances where the total number of machines which need to access the Internet exceeds the supply of IP addresses.

Any attempt to perform NAT operations on IPsec packets *between the IPsec gateways* creates a basic conflict:

- IPsec wants to authenticate packets and ensure they are unaltered on a gateway-to-gateway basis
- NAT rewrites packet headers as they go by
- IPsec authentication fails if packets are rewritten anywhere between the IPsec gateways

For IKE and ESP it is not necessarily fatal, but is certainly an unwelcome complication.

NAT on or behind the IPsec gateway works

This problem can be avoided by having the masquerading take place *on or behind* the IPsec gateway.

This can be done physically with two machines, one physically behind the other. A picture, using SG to indicate IPsec **S** ecurity **G**ateways, is:

```
clients --- NAT ----- SG ---------- SG
            two machines
```

In this configuration, the actual client addresses need not be given in the *leftsubnet=* parameter of the FreeS/WAN connection description. The security gateway just delivers packets to the NAT box; it needs only that machine's address. What that machine does with them does not affect FreeS/WAN.

A more common setup has one machine performing both functions:

```
clients ----- NAT/SG ---------------SG
            one machine
```

Here you have a choice of techniques depending on whether you want to make your client subnet visible to clients on the other end:

- If you want the single gateway to behave like the two shown above, with your clients hidden behind the NAT, then omit the *leftsubnet=* parameter. It then defaults to the gateway address. Clients on the other end then talk via the tunnel only to your gateway. The gateway takes packets emerging from the tunnel, applies normal masquerading, and forwards them to clients.

- If you want to make your client machines visible, then give the client subnet addresses as the *leftsubnet=* parameter in the connection description and
  either
  set *leftfirewall=yes* to use the default *updown* script
  or
  use your own script by giving its name in a *leftupdown=* parameter
  These scripts are described in their own section.

  In this case, no masquerading is done. Packets to or from the client subnet are encrypted or decrypted without any change to their client subnet addresses, although of course the encapsulating packets use gateway addresses in their headers. Clients behind the right security gateway see a route via that gateway to the left subnet.

NAT between gateways is problematic

We recommend not trying to build IPsec connections which pass through a NAT machine. This setup poses problems:

```
clients --- SG --- NAT ---------- SG
```

If you must try it, some references are:

- Jean_Francois Nadeau's document on doing IPsec behind NAT
- VPN masquerade patches to make a Linux NAT box handle IPsec packets correctly

Other references on NAT and IPsec

Other documents which may be relevant include:

- an Internet Draft on IPsec and NAT which may eventually evolve into a standard solution for this problem.
- an informational RFC, *Security Model with Tunnel-mode IPsec for NAT Domains* .
- an article in Cisco's *Internet Protocol Journal*

Other complications

Of course simply allowing UDP 500 and ESP packets is not the whole story. Various other issues arise in making IPsec and packet filters co-exist and even co-operate. Some of them are summarised below.

IPsec *through* the gateway

Basic IPsec packet filtering rules deal only with packets addressed to or sent from your IPsec gateway.

It is a separate policy decision whether to permit such packets to pass through the gateway so that client machines can build end-to-end IPsec tunnels of their own. This may not be practical if you are using NAT (IP masquerade) on your gateway, and may conflict with some corporate security policies.

Where possible, allowing this is almost certainly a good idea. Using IPsec on an end-to-end basis is more secure than gateway-to-gateway.

Doing it is quite simple. You just need firewall rules that allow UDP port 500 and protocols 50 and 51 to pass through your gateway. If you wish, you can of course restrict this to certain hosts.

Preventing non-IPsec traffic

You can also filter *everything but* UDP port 500 and ESP to restrict traffic to IPsec only, either for anyone communicating with your host or just for specific partners.

One application of this is for the telecommuter who might have:

```
Sunset==========West-----------------East ================ firewall --- th
    home network      untrusted net        corporate network
```

The subnet on the right is 0.0.0.0/0, the whole Internet. The West gateway is set up so that it allows only IPsec packets to East in or out.

This configuration is used in AT&T Research's network. For details, see the papers links in our introduction.

Another application would be to set up firewall rules so that an internal machine, such as an employees-only web server, could not talk to the outside world except via specific IPsec tunnels.

Filtering packets from unknown gateways

It is possible to use firewall rules to restrict UDP 500 and ESP packets so that these packets are accepted only from known gateways. This is not strictly necessary since FreeS/WAN will discard packets from unknown gateways. You might, however, want to do it for any of a number of reasons. For example:

- Arguably, "belt and suspenders" is the sensible approach to security. If you can block a potential attack in two ways, use both. The only question is whether to look for a third way after implementing the first two.
- Some admins may prefer to use the firewall code this way because they prefer firewall logging to FreeS/WAN's logging.
- You may need it to implement your security policy. Consider an employee working at home, and a policy that says traffic from the home system to the Internet at large must go first via IPsec to the corporate LAN and then out to the Internet via the corporate firewall. One way to do that is to make *ipsec0* the default route on the home gateway and provide exceptions only for UDP 500 and ESP to the corporate gateway. Everything else is then routed via the tunnel to the corporate gateway.

It is not possible to use only static firewall rules for this filtering if you do not know the other gateways' IP addresses in advance, for example if you have "road warriors" who may connect from a different address each time or if want to do opportunistic encryption to arbitrary gateways. In these cases, you can accept UDP 500 IKE packets from anywhere, then use the updown script feature of pluto(8) to dynamically adjust firewalling for each negotiated tunnel.

Firewall packet filtering does not much reduce the risk of a denial of service attack on FreeS/WAN. The firewall can drop packets from unknown gateways, but KLIPS does that quite efficiently anyway, so you gain little. The firewall cannot drop otherwise legitmate packets that fail KLIPS authentication, so it cannot protect against an attack designed to exhaust resources by making FreeS/WAN perform many expensive authentication operations.

In summary, firewall filtering of IPsec packets from unknown gateways is possible but not strictly

necessary.

Other packet filters

When the IPsec gateway is also acting as your firewall, other packet filtering rules will be in play. In general, those are outside the scope of this document. See our Linux firewall links for information. There are a few types of packet, however, which can affect the operation of FreeS/WAN or of diagnostic tools commonly used with it. These are discussed below.

ICMP filtering

ICMP is the **I** nternet **C**ontrol **M**essage **P** rotocol. It is used for messages between IP implementations themselves, whereas IP used is used between the clients of those implementations. ICMP is, unsurprisingly, used for control messages. For example, it is used to notify a sender that a desination is not reachable, or to tell a router to reroute certain packets elsewhere.

ICMP handling is tricky for firewalls.

- You definitely want some ICMP messages to get through; things won't work without them. For example, your clients need to know if some destination they ask for is unreachable.
- On the other hand, you do equally definitely do not want untrusted folk sending arbitrary control messages to your machines. Imagine what someone moderately clever and moderately malicious could do to you, given control of your network's routing.

ICMP does not use ports. Messages are distinguished by a "message type" field and, for some types, by an additional "code" field. The definitive list of types and codes is on the IANA site.

One expert uses this definition for ICMP message types to be dropped at the firewall.

```
# ICMP types which lack socially redeeming value.
#  5     Redirect
#  9     Router Advertisement
# 10     Router Selection
# 15     Information Request
# 16     Information Reply
# 17     Address Mask Request
# 18     Address Mask Reply

badicmp='5 9 10 15 16 17 18'
```

A more conservative approach would be to make a list of allowed types and drop everything else.

Whichever way you do it, your ICMP filtering rules on a FreeS/WAN gateway should allow at least the following ICMP packet types:

echo (type 8)
echo reply (type 0)
: These are used by ping(1). We recommend allowing both types through the tunnel and to or from your gateway's external interface, since ping(1) is an essential testing tool.

  It is fairly common for firewalls to drop ICMP echo packets addressed to machines behind the firewall. If that is your policy, please create an exception for such packets arriving via an IPsec tunnel, at least during intial testing of those tunnels.

destination unreachable (type 3)
: This is used, with code 4 (Fragmentation Needed and Don't Fragment was Set) in the code field, to control path MTU discovery. Since IPsec processing adds headers, enlarges packets and may cause fragmentation, an IPsec gateway should be able to send and receive these ICMP messages **on both inside and outside interfaces**.

UDP packets for traceroute

The traceroute(1) utility uses UDP port numbers from 33434 to approximately 33633. Generally, these should be allowed through for troubleshooting.

Some firewalls drop these packets to prevent outsiders exploring the protected network with traceroute(1). If that is your policy, consider creating an exception for such packets arriving via an IPsec tunnel, at least during intial testing of those tunnels.

UDP for L2TP

Windows 2000 does, and products designed for compatibility with it may, build L2TP tunnels over IPsec connections.

For this to work, you must allow UDP protocol 1701 packets coming out of your tunnels to continue to their destination. You can, and probably should, block such packets to or from your external interfaces, but allow them from *ipsec0*.

See also our Windows 2000 interoperation discussion.

How it all works: IPsec packet details

IPsec uses two main types of packet:

IKE uses **the UDP protocol and port 500**.
: Unless you are using only (less secure, not recommended) manual keying, you need IKE to negotiate connection parameters, acceptable algorithms, key sizes and key setup. IKE handles everything required to set up, rekey, repair or tear down IPsec connections.

ESP is **protocol number 50**
: This is required for encrypted connections.

All of those packets should have appropriate IPsec gateway addresses in both the to and from IP header fields. Firewall rules can check this if you wish, though it is not strictly necessary. This is discussed in more detail later.

IPsec processing of incoming packets authenticates them then removes the ESP header and decrypts if necessary. Successful processing exposes an inner packet which is then delivered back to the firewall machinery, marked as having arrived on an *ipsec[0-3]* interface. Firewall rules can use that interface label to distinguish these packets from unencrypted packets which are labelled with the physical interface they arrived on (or perhaps with a non-IPsec virtual interface such as *ppp0*).

One of our users sent a mailing list message with a diagram of the packet flow.

ESP does not have ports

Some protocols, such as TCP and UDP, have the notion of ports. Other protocols, including ESP, do not. Quite a few IPsec newcomers have become confused on this point. There are no ports *in* the ESP

protocol, and no ports used *for* it. For this protocols, *the idea of ports is completely irrelevant*.

Header layout

The protocol number for ESP is used in the 'next header' field of the IP header. On most non-IPsec packets, that field would have one of:

- 1 for ICMP
- 4 for IP-in-IP encapsulation
- 6 for TCP
- 17 for UDP
- ... or one of about 100 other possibilities listed by IANA

Each header in the sequence tells what the next header will be. IPsec adds headers for ESP near the beginning of the sequence. The original headers are kept and the 'next header' fields adjusted so that all headers can be correctly interpreted.

For example, using **[ ]** to indicate data protected by ESP and unintelligible to an eavesdropper between the gateways:

- a simple packet might have only IP and TCP headers with:
  - IP header says next header --> TCP
  - TCP header port number --> which process to send data to
  - data
- with ESP transport mode encapsulation, that packet would have:
  - IP header says next header --> ESP
  - ESP header **[** says next --> TCP
  - TCP header port number --> which process to send data to
  - data **]**

  Note that the IP header is outside ESP protection, visible to an attacker, and that the final destination must be the gateway.
- with ESP in tunnel mode, we might have:
  - IP header says next header --> ESP
  - ESP header **[** says next --> IP
  - IP header says next header --> TCP
  - TCP header port number --> which process to send data to
  - data **]**

  Here the inner IP header is protected by ESP, unreadable by an attacker. Also, the inner header can have a different IP address than the outer IP header, so the decrypted packet can be routed from the IPsec gateway to a final destination which may be another machine.

Part of the ESP header itself is encrypted, which is why the **[** indicating protected data appears in the middle of some lines above. The next header field of the ESP header is protected. This makes traffic analysis more difficult. The next header field would tell an eavesdropper whether your packet was UDP to the gateway, TCP to the gateway, or encapsulated IP. It is better not to give this information away. A clever attacker may deduce some of it from the pattern of packet sizes and timings, but we need not make it easy.

IPsec allows various combinations of these to match local policies, including combinations that nest multiple copies of these headers.

For example, suppose my employer has an IPsec VPN running between two offices so all packets travelling between the gateways for those offices are encrypted. If gateway policies allow it (The admins could block UDP 500 and protocols 50 and 51 to disallow it), I can build an IPsec tunnel

from my desktop to a machine in some remote office. Those packets will have one ESP header throughout their life, for my end-to-end tunnel. For part of the route, however, they will also have another ESP layer for the corporate VPN's encapsulation. The whole header scheme for a packet on the Internet might be:

- IP header (with gateway address) says next header --> ESP
- ESP header **[** says next --> IP
- IP header (with receiving machine address) says next header --> ESP
- ESP header **[** says next --> TCP
- TCP header port number --> which process to send data to
- data **]]**

The first ESP (outermost) header is for the corporate VPN. The inner ESP header is for the secure machine-to-machine link.

DHR on the updown script

Here are some mailing list comments from pluto(8) developer Hugh Redelmeier on an earlier draft of this document:

```
There are many important things left out

- firewalling is important but must reflect (implement) policy.  Since
  policy isn't the same for all our customers, and we're not experts,
  we should concentrate on FW and MASQ interactions with FreeS/WAN.

- we need a diagram to show packet flow WITHIN ONE MACHINE, assuming
  IKE, IPsec, FW, and MASQ are all done on that machine.  The flow is
  obvious if the components are run on different machines (trace the
  cables).

  IKE input:
        + packet appears on public IF, as UDP port 500
        + input firewalling rules are applied (may discard)
        + Pluto sees the packet.

  IKE output:
        + Pluto generates the packet & writes to public IF, UDP port 500
        + output firewalling rules are applied (may discard)
        + packet sent out public IF

  IPsec input, with encapsulated packet, outer destination of this host:
        + packet appears on public IF, protocol 50 or 51.  If this
          packet is the result of decapsulation, it will appear
          instead on the paired ipsec IF.
        + input firewalling rules are applied (but packet is opaque)
        + KLIPS decapsulates it, writes result to paired ipsec IF
        + input firewalling rules are applied to resulting packet
          as input on ipsec IF
        + if the destination of the packet is this machine, the
          packet is passed on to the appropriate protocol handler.
          If the original packet was encapsulated more than once
          and the new outer destination is this machine, that
          handler will be KLIPS.
        + otherwise:
          * routing is done for the resulting packet.  This may well
            direct it into KLIPS for encoding or encrypting.  What
            happens then is described elsewhere.
          * forwarding firewalling rules are applied
          * output firewalling rules are applied
```

```
          * the packet is sent where routing specified

 IPsec input, with encapsulated packet, outer destination of another host:
        + packet appears on some IF, protocol 50 or 51
        + input firewalling rules are applied (but packet is opaque)
        + routing selects where to send the packet
        + forwarding firewalling rules are applied (but packet is opaque)
        + packet forwarded, still encapsulated

  IPsec output, from this host or from a client:
        + if from a client, input firewalling rules are applied as the
          packet arrives on the private IF
        + routing directs the packet to an ipsec IF (this is how the
          system decides KLIPS processing is required)
        + if from a client, forwarding firewalling rules are applied
        + KLIPS eroute mechanism matches the source and destination
          to registered eroutes, yielding a SPI group.  This dictates
          processing, and where the resulting packet is to be sent
          (the destinations SG and the nexthop).
        + output firewalling is not applied to the resulting
          encapsulated packet

- Until quite recently, KLIPS would double encapsulate packets that
  didn't strictly need to be.  Firewalling should be prepared for
  those packets showing up as ESP protocol input packets on
  an ipsec IF.

- MASQ processing seems to be done as if it were part of the
  forwarding firewall processing (this should be verified).

- If a firewall is being used, it is likely the case that it needs to
  be adjusted whenever IPsec SAs are added or removed.  Pluto invokes
  a script to do this (and to adjust routing) at suitable times.  The
  default script is only suitable for ipfwadm-managed firewalls.  Under
  LINUX 2.2.x kernels, ipchains can be managed by ipfwadm (emulation),
  but ipchains more powerful if manipulated using the ipchains command.
  In this case, a custom updown script must be used.

  We think that the flexibility of ipchains precludes us supplying an
  updown script that would be widely appropriate.
```

Contents Previous Next