Global Information Assurance Certification Paper

IPSEC VPN using FreeBSD
Greg Panula
GSEC Practical version 1.2e

Introduction

25 July 2001

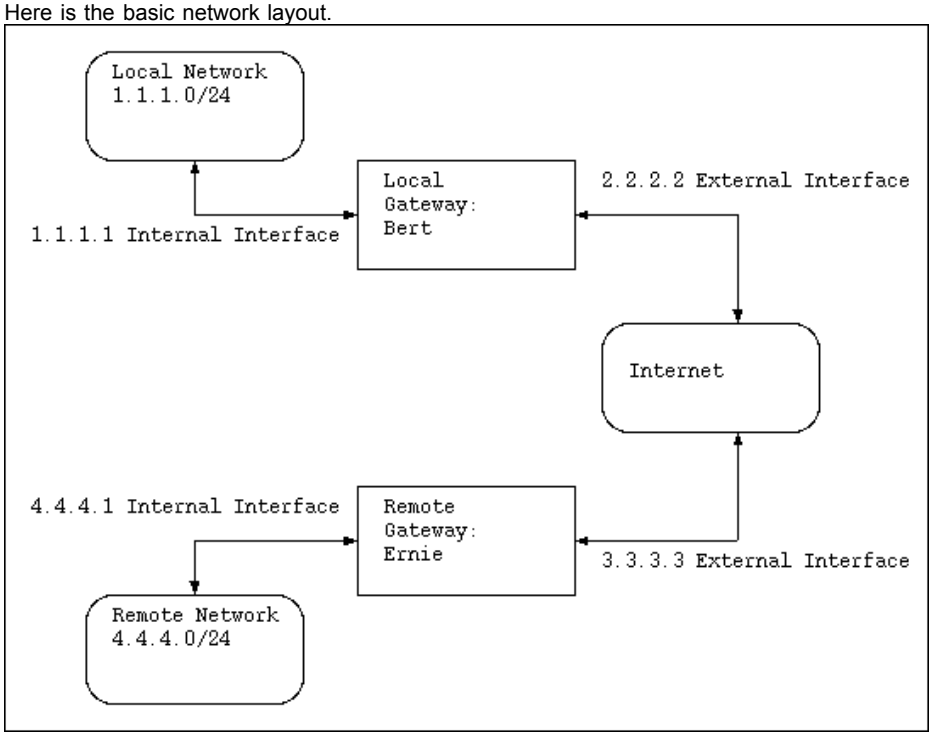
Virtual Private Networks(VPNs) are becoming more common every day. Most of the documention/how-to's out there cover single user VPN connectivity or LAN-to-LAN VPN connectivity. Most of these solutions don't provide a way to NAT one or both ends of the traffic. The ability to NAT this traffic will grow in importance as more business to business(B2B) VPN connections are built and network numbering conflicts occur. Network numbering conflicts are more likely to occur because of private networks using the RFC 1918 network space.

This paper will demostrate a way to setup an IPSec VPN that will allow for NAT'ing using FreeBSD boxes as the gateway machines. It also has the bonus of being a fairly easy method for connecting WANs across public networks. The information and examples provided here should be compatible with other open-source unixes

The items covered in this paper are: setting up the tunnel using gif interfaces, ipsec to encrypt the traffic, racoon for automatic key exchange, setting up some simple firewalling and setting up some simple NAT.

Network Layout

Here is the basic network layout.



The initial goal is to have the two gateways automatically route and encrypt traffic between the two networks. I'll demostrate the NAT'ing after completing the initial goal.

Initial Tunnel Configuration

The first step was to setup a tunnel between the two gateways.

This is acomplished by using gifconfig to configure a generic IP tunnel. You need gif compiled in your kernel. The Generic kernel comes with 4 gif interfaces. Here is kernel config line from the Generic kernel:

```
pseudo-device    gif    4    # IPv6 and IPv4 tunneling
```

Here is the man page for gifconfig
Here is the man page for gif

To make firewalling and managing traffic flowing thru the ip tunnel a little easier I used virtual interfaces; I added aliases to the loopback interface(lo0) on both gateways to use as inside end-points for the tunnel. That way I have a chance to control the traffic at the gateway before passing it on out the internal interface to it's local network. Useful for NAT situations, trouble-shooting and easier to setup firewall rules because it is easier to picture/diagram the network flow.

First setup the aliases
On bert I added 5.5.5.1 as the alias

ifconfig lo0 alias 5.5.5.1 netmask 255.255.255.252

On ernie I added 5.5.5.2 as the alias

ifconfig lo0 alias 5.5.5.2 netmask 255.255.255.252

The .252 netmask is a subnet with only two hosts in it. I figured this would help me keep clear who was connected to who. You can use whatever netmask you like.

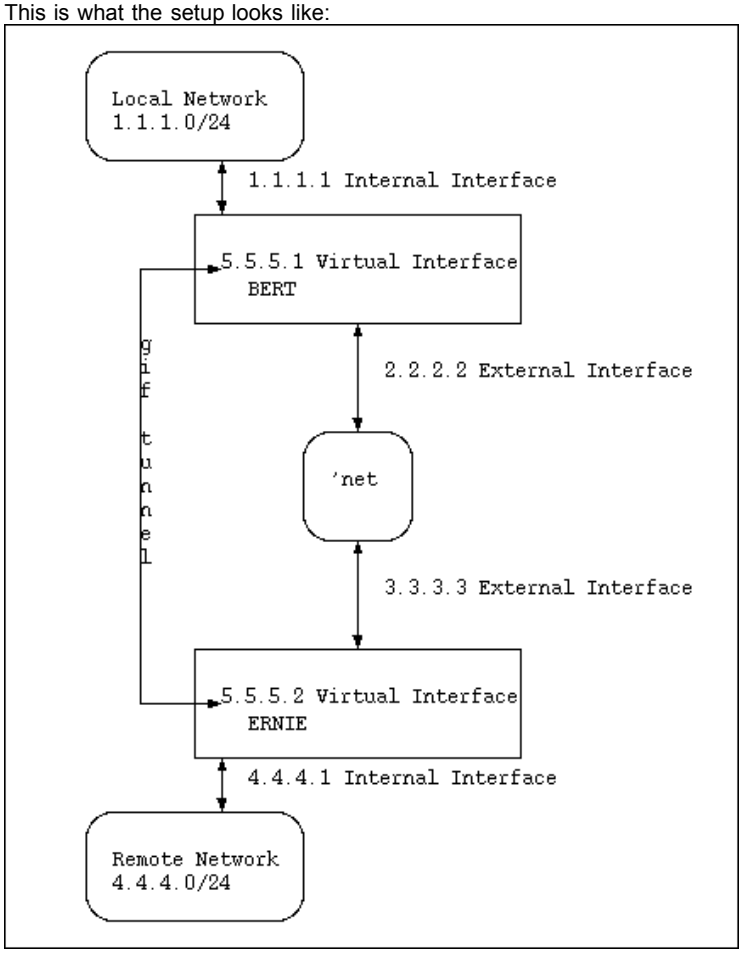
Next actually setup the tunnel
On bert I did this:

gifconfig gif0 2.2.2.2 3.3.3.3
ifconfig gif0 inet 5.5.5.1 5.5.5.2 netmask 255.255.255.252

On ernie I did this:

gifconfig gif0 3.3.3.3 2.2.2.2
ifconfig gif0 inet 5.5.5.2 5.5.5.1 netmask 255.255.255.252

This is what the setup looks like:



The gif tunnels external end-points are 2.2.2.2 and 3.3.3.3 and the internal end-points are 5.5.5.1 and 5.5.5.2.

Quick Connectivity Check

At this point a quick ping from ernie to bert's 5.5.5.1 address to confirm the tunnel is indeed up and passing traffic.
On ernie: ping 5.5.5.1

tcpdump view on bert's external interface:

3.3.3.3 > 2.2.2.2: 5.5.5.2 > 5.5.5.1: icmp: echo request (ipip)
2.2.2.2 > 3.3.3.3: 5.5.5.1 > 5.5.5.2: icmp: echo reply (ipip)
3.3.3.3 > 2.2.2.2: 5.5.5.2 > 5.5.5.1: icmp: echo request (ipip)
2.2.2.2 > 3.3.3.3: 5.5.5.1 > 5.5.5.2: icmp: echo reply (ipip)

The tunnel is up and passing traffic. Now to add routes to bert & ernie so they are aware of each other's local network. RIP or another routing protocol would be a cleaner solution than static routes.
On bert

route add 4.4.4.0/24 5.5.5.2

On ernie

route add 1.1.1.0/24 5.5.5.1

And then a quick ping from a host on the 4.4.4.0/24 network to a host on the 1.1.1.0/24 network.
On host 4.4.4.4

ping 1.1.1.10

tcpdump view on bert's external interface:

3.3.3.3 > 2.2.2.2: 4.4.4.4 > 1.1.1.10: icmp: echo request (ipip)
2.2.2.2 > 3.3.3.3: 1.1.1.10 > 4.4.4.4: icmp: echo reply (ipip)
3.3.3.3 > 2.2.2.2: 4.4.4.4 > 1.1.1.10: icmp: echo request (ipip)
2.2.2.2 > 3.3.3.3: 1.1.1.10 > 4.4.4.4: icmp: echo reply (ipip)

Traffic is flowing between the two networks but isn't being encrypted. The contents are plainly visible; IP in IP tunnel, host 4.4.4.4 is pinging host 1.1.1.10 and getting replies back.

IPSec Policy Setup

Next was setting up IPSec policies to tell the gateways what traffic flows I wanted encrypted.

The traffic flow I want encrypted is any and all traffic between the two gateways(bert & ernie).

At this point you'll **need** a secondary way to access the remote gateway(ernie), as it will want all traffic coming from the local gateway(bert) to be encrypted. Secure shell from another network/gateway or dial-up access are good secondary access methods.

setkey is used for defining ipsec policies on a FreeBSD box. Here is the man page for setkey.

On bert I setup two policies, one for traffic going from himself to ernie and another one for traffic coming ernie to him.

```
setkey -c << EOF
spdadd 2.2.2.2 3.3.3.3 any -P out ipsec esp/tunnel/2.2.2.2-3.3.3.3/require;
spdadd 3.3.3.3 2.2.2.2 any -P in ipsec esp/tunnel/3.3.3.3-2.2.2.2/require;
EOF
```

On ernie, same thing as bert; one policy for traffic going from ernie to bert and another for the return traffic.

```
setkey -c << EOF
spdadd 3.3.3.3 2.2.2.2 any -P out ipsec esp/tunnel/3.3.3.3-2.2.2.2/require;
spdadd 2.2.2.2 3.3.3.3 any -P in ipsec esp/tunnel/2.2.2.2-3.3.3.3/require;
EOF
```

The policies above will encrypt any traffic flowing between the external interfaces of Bert&Ernie. And by using tunnel mode it will also provide protection("authenication") of the IP packet. The tunneled packeted in this case is the the ip-in-ip tunnel created by the gif0 interface.

By using tunnel mode between the external interfaces of the gateways the payload of the packets and the original IP header is encrypted into a single payload with a new IP header added to the outgoing packet. Someone looking at traffic will be unable to determine it is an IP in IP tunnel. They will only see encrypted traffic flowing between Bert and Ernie. It also assures that the kernel processes the tunnel and IPSec in the correct order. Arriving encrypted traffic is first decrypted at the external interface, then the external interface processes the IP in IP tunnel(gif tunnel) which passes the internal IP packet to the internal end-point of the gif tunnel (the virtual interface) for delivery to its destination(internal private network).

Here is a paragraph from rfc2041 that might help.

> For a tunnel mode SA, there is an "outer" IP header that specifies the IPsec processing destination, plus an "inner" IP header that specifies the (apparently) ultimate destination for the packet. The security protocol header appears after the outer IP header, and before the inner IP header. If AH is employed in tunnel mode, portions of the outer IP header are afforded protection (as above), as well as all of the tunneled IP packet (i.e., all of the inner IP header is protected, as well as higher layer protocols). If ESP is employed, the protection is afforded only to the tunneled packet, not to the outer header.

Here is some more useful info taken from rfc2406.

Tunnel mode ESP may be employed in either hosts or security gateways. When ESP is implemented in a security gateway (to protect subscriber transit traffic), tunnel mode must be used. In tunnel mode, the "inner" IP header carries the ultimate source and destination addresses, while an "outer" IP header may contain distinct IP addresses, e.g., addresses of security gateways. In tunnel mode, ESP protects the entire inner IP packet, including the entire inner IP header. The position of ESP in tunnel mode, relative to the outer IP header, is the same as for ESP in transport mode. The following diagram illustrates ESP tunnel mode positioning for typical IPv4 and IPv6 packets.

```
------------------------------------------------------------
IPv4 | new IP hdr* | | orig IP hdr* | | | ESP | ESP|
|(any options)| ESP | (any options) |TCP|Data|Trailer|Auth|
------------------------------------------------------------
|<--------- encrypted ---------->|
|<---------- authenticated ---------->|

------------------------------------------------------------
IPv6 | new* |new ext | | orig*|orig ext | | | ESP | ESP|
|IP hdr| hdrs* |ESP|IP hdr| hdrs * |TCP|Data|Trailer|Auth|
------------------------------------------------------------
|<--------- encrypted ---------->|
|<---------- authenticated ---------->|
```

Setup automatic key generation and exchange

Automatic key generation and exchange between two hosts is done in two phases. The first phase (IKE phase 1) is used to authenticate each other and then to agree on an encryption algorithm for exchanging keys. After phase 1 completes the two hosts agree on an encryption algorithm to use for encrypting data, what traffic to encrypt, generate keys and then exchange keys (IKE phase 2). The keys are first encrypted with the algorithm agreed upon in phase 1 using either a pre-shared secret key or X.509 certificate as a seed.

Usually two algorithms are used for encrypting traffic streams; an authentication algorithm and an encryption algorithm. The authentication algorithm provides a hash to protect against tampering and the encryption algorithm is a strong encryption algorithm for securing the data/payload. The authentication algorithm is either MD5 or SHA1; 128-bit or 160-bit key lengths. The strong encryption key lengths range from 64-bit to 2040-bit, depending on algorithm used. The IPSec package for FreeBSD currently supports the following strong encryption algorithms; des, 3des, blowfish, cast128, and rc5.

To handle the automatic key generation and exchange I used racoon. There is a port for it and information about it is available here . The version I used was 20010222a. The most recent version in the ports collection is currently 20010418a. I recommend using the most recent version.

After building and installing the port, you'll need to create a config file for it. Luckily the port comes with a basic config file.

```
cd /usr/local/etc/racoon
cp racoon.conf.dist racoon.dist
```

Now edit the racoon.conf for our setup. Use your least favorite text editor.
Under the listen section add a line for the external interface
On bert the added line is:

```
isakmp 2.2.2.2 [500]
```

On ernie the added line is:

```
isakmp 3.3.3.3 [500]
```

The listen section tells the daemon what ip address and port to bind to. If you don't specify anything under the Listen section, the daemon will attempt to bind to port 500 on all available interfaces.

Next is setting up the encryption algorithms to use, key life times and SAs.
In bert's racoon.conf I added the following:

```
remote 3.3.3.3 [500]
{
exchange_mode aggressive,main;
doi ipsec_doi;
situation identity_only;
nonce_size 16;
lifetime time 1 min; # sec,min,hour
lifetime byte 5 MB; # B,KB,GB
```

```
initial_contact on;
support_mip6 on;
proposal_check obey; # obey, strict or claim
proposal {
encryption_algorithm blowfish;
hash_algorithm sha1;
authentication_method pre_shared_key ;
dh_group 2 ;
}
}

sainfo address 2.2.2.2 any address 3.3.3.3 any
{
pfs_group 1;
lifetime time 3600 sec;
lifetime byte 50 MB;
encryption_algorithm blowfish;
authentication_algorithm hmac_sha1;
compression_algorithm deflate;
}
```

The remote section is for IKE phase 1 and the sainfo section is for IKE phase 2.

This configures racoon on Bert to use blowfish for the encryption algorithm, key lifetimes are 1 hour or 50mb worth of traffic which ever comes first. And there is the directive to use a pre-shared secret for IKE phase 1.

In ernie's racoon.conf I added the following

```
remote 2.2.2.2 [500]
{
exchange_mode aggressive,main;
doi ipsec_doi;
situation identity_only;
nonce_size 16;
lifetime time 1 min; # sec,min,hour
lifetime byte 5 MB; # B,KB,GB
initial_contact on;
support_mip6 on;
proposal_check obey; # obey, strict or claim
proposal {
encryption_algorithm blowfish;
hash_algorithm sha1;
authentication_method pre_shared_key ;
dh_group 2 ;
}
}
sainfo address 3.3.3.3 any address 2.2.2.2 any
{
pfs_group 1;
lifetime time 3600 sec;
lifetime byte 50 MB;
encryption_algorithm blowfish;
authentication_algorithm hmac_sha1;
compression_algorithm deflate ;
}
```

Basically the same setup as bert but in reverse.

Since I don't have much experience with X.509 certificates I decided to go with a pre-shared secret. The file /usr/local/etc/racoon/psk.txt is the pre-shared secret file listed in the default racoon.conf. Contents of the file are very straight forward. It is simply:
peer_ip_address sharedkey

On bert I did this

```
cd /usr/local/etc/racoon
echo 3.3.3.3 blah@blah@blah > psk.txt
chmod 600 psk.txt
```

On ernie I did this

```
cd /usr/local/etc/racoon
echo 2.2.2.2 blah@blah@blah > psk.txt
chmod 600 psk.txt
```

The psk.txt file must be read-writable by only root. Otherwise racoon won't run. It is also a good security measure.

Test IPSec connectivity

Now we can test the encrypted tunnel.

First off double-check the ipsec policy on both gateways.
On bert:

```
setkey -DP

 spits out the current policies

3.3.3.3[any] 2.2.2.2[any] any
in ipsec
esp/tunnel/3.3.3.3-2.2.2.2/require
spid=4 seq=1 pid=18477
refcnt=1
2.2.2.2[any] 3.3.3.3[any] any
out ipsec
esp/tunnel/2.2.2.2-3.3.3.3/require
spid=3 seq=0 pid=18477
refcnt=1

 On ernie:

setkey -DP

 spits out the current policies

2.2.2.2[any] 3.3.3.3[any] any
in ipsec
esp/tunnel/2.2.2.2-3.3.3.3/require
spid=2 seq=1 pid=21277
refcnt=1
3.3.3.3[any] 2.2.2.2[any] any
out ipsec
esp/tunnel/3.3.3.3-2.2.2.2/require
spid=1 seq=0 pid=21277
refcnt=1
```

A "setkey -D" won't return anything because there hasn't been an exchange of keys, yet.

Then startup racoon on both gateways; /usr/local/sbin/racoon . You might want to start it up in foreground mode the first time out;
/usr/local/sbin/racoon -F

You should see something like this when starting up racoon in foreground mode:

```
INFO: main.c:141:main(): @(#)racoon 20010222 sakane@ydc.co.jp
INFO: main.c:142:main(): @(#)This product linked software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)
WARNING: pfkey.c:1940:pk_checkalg(): compression algorithm can not be checked.
INFO: isakmp.c:1262:isakmp_open(): 2.2.2.2[500] used as isakmp port (fd=6)
```

Next just a simple ping from bert to ernie's 5.5.5.2 address.
On ernie

```
ping 5.5.5.2
```

At this point racoon should be doing something. You should see some messages like this scroll by:

```
INFO: isakmp.c:1586:isakmp_post_acquire(): IPsec-SA request for 3.3.3.3 queued due to no phase1 found.
INFO: isakmp.c:771:isakmp_ph1begin_i(): initiate new phase 1 negotiation: 2.2.2.2[500]<=>3.3.3.3[500]
INFO: isakmp.c:776:isakmp_ph1begin_i(): begin Aggressive mode.
INFO: vendorid.c:91:check_vendorid(): Vendor ID matched.
INFO: isakmp.c:2301:log_ph1established(): ISAKMP-SA established 2.2.2.2[500]-3.3.3.3[500] spi:7cd55bc24955eae1:5840c9bcf44f798e
INFO: pfkey.c:1113:pk_recvupdate(): IPsec-SA established: ESP/Tunnel 3.3.3.3->2.2.2.2 spi=212240116(0x87ddfd0)
INFO: pfkey.c:1299:pk_recvadd(): IPsec-SA established: ESP/Tunnel 2.2.2.2->3.3.3.3 spi=232595730(0x567731b)
```

The first few ping requests are lost because it takes a couple of seconds for the inital key exchange to happen.

Here is what the tcpdump from bert's external interface shows:

```
2.2.2.2.500 > 3.3.3.3.500: isakmp: phase 1 I agg: [|sa]
3.3.3.3.500 > 2.2.2.2.500: isakmp: phase 1 R agg: [|sa]
2.2.2.2.500 > 3.3.3.3.500: isakmp: phase 1 I agg: (hash: len=20)
```

```
2.2.2.2.500 > 3.3.3.3.500: isakmp: phase 2/others I inf[E]: [|hash]
2.2.2.2.500 > 3.3.3.3.500: isakmp: phase 2/others I oakley-quick[E]: [|hash]
3.3.3.3.500 > 2.2.2.2.500: isakmp: phase 2/others R oakley-quick[E]: [|hash]
2.2.2.2.500 > 3.3.3.3.500: isakmp: phase 2/others I oakley-quick[E]: [|hash]
2.2.2.2 > 3.3.3.3: ESP(spi=232595730,seq=0x1)
3.3.3.3 > 2.2.2.2: ESP(spi=212240116,seq=0x1)
2.2.2.2 > 3.3.3.3: ESP(spi=232595730,seq=0x2)
3.3.3.3 > 2.2.2.2: ESP(spi=212240116,seq=0x2)
2.2.2.2 > 3.3.3.3: ESP(spi=232595730,seq=0x3)
3.3.3.3 > 2.2.2.2: ESP(spi=212240116,seq=0x3)
2.2.2.2 > 3.3.3.3: ESP(spi=232595730,seq=0x4)
3.3.3.3 > 2.2.2.2: ESP(spi=212240116,seq=0x4)
```

Identities were confirmed, keys exchanged and traffic is encrypted and flowing

Now if you do a "setkey -D" you'll get an output like this:

```
2.2.2.2 3.3.3.3
esp mode=tunnel spi=232595730(0x0ddd2112) reqid=0(0x00000000)
E: blowfish-cbc f4820d3d 5783d340 dffc23dd 06a6cae1
A: hmac-sha1 803226d3 38564194 3913f9ac 864e25db bc08440d
replay=4 flags=0x00000000 state=mature seq=1 pid=18513
created: Apr 23 08:39:51 2001 current: Apr 23 08:41:29 2001
diff: 98(s) hard: 3600(s) soft: 2880(s)
last: Apr 23 08:40:15 2001 hard: 0(s) soft: 0(s)
current: 1120(bytes) hard: 52428800(bytes) soft: 41943040(bytes)
allocated: 7 hard: 0 soft: 0
refcnt=2
3.3.3.3 2.2.2.2
esp mode=tunnel spi=212240116(0x0ca686f4) reqid=0(0x00000000)
E: blowfish-cbc ebf31bca d3db156c 73337895 fb2f6ef6
A: hmac-sha1 dd6e54e2 2a757942 d056ee6b 7fcf6f95 2aa1eac0
replay=4 flags=0x00000000 state=mature seq=0 pid=18513
created: Apr 23 08:39:51 2001 current: Apr 23 08:41:29 2001
diff: 98(s) hard: 3600(s) soft: 2880(s)
last: Apr 23 08:40:15 2001 hard: 0(s) soft: 0(s)
current: 728(bytes) hard: 52428800(bytes) soft: 41943040(bytes)
allocated: 7 hard: 0 soft: 0
refcnt=1
```

When the a key's lifetime approaches a soft limit racoon will generate a fresh set of keys to use and once those keys are generated it will start using them. That way you don't lose those couple of packets during key exchange, like when we first initalized the encrypted tunnel.

In my limited stress testing, I found a P166 could handle an ipsec tunnel passing a continous stream of traffic at 25KB/sec, as long as there wasn't a lot of fragmention. With large amounts of fragmention latency increased but packets still arrived.

Fragmention would happen when a host on one of the internal networks sent large packets(approximately equal to the MTU of the external interface of the gateway). The fragmention usually happened because adding the "outer" ESP header caused the packet to become larger than the Maximum Transmit Unit of the external interface of the gateway. Which forced the gateway to break the packet into two parts. Remember in tunnel mode the original packet is encrypted and a new ip header is added. This makes going over the MTU rather easy.

If latency and/or cpu usage on the gateway are concerns and there is a server on one network that will be sending a large amount of data to the remote network, you might consider reducing the sending server's MTU by 100 bytes. The sending server will generate more packets but the packets will be the perfect size for adding the ipsec header to, i.e. They won't need to be broken in two by the gateway thus reducing latency and cpu usage on the gateway machines.

Sample Firewall Rules

In this section I will show a sample firewall configuration using IPFW as the firewall tool. IPFW is a first-match firewall tool. Please read this section in the FreeBSD handbook for information about IPFW.

On Bert we have the following interfaces:
fxp0 2.2.2.2 external interface
fxp1 1.1.1.1 internal interface
gif0 gif tunnel interface
lo0 5.5.5.1 virtual interface / alias on loopback interface

On Ernie we have the following interfaces:
fxp0 3.3.3.3 external interface
fxp1 4.4.4.1 internal interface
gif0 gif tunnel interface
lo0 5.5.5.2 virtual interface / alias on loopback interface

Bert's internal network is 1.1.1.0/24
Ernie's internal network is 4.4.4.0/24

Firewall goals:
Allow only ipsec traffic between Bert & Ernie's external interfaces
I want to allow traffic from Bert's internal network to inititate contact with Ernie's internal network, but don't want Ernie's network to be able to initate contact with Bert's internal network.
I don't want to allow NetBIOS traffic from either network to reach the other.
I any want to allow ssh, imap, http and PCAnywhere traffic from Bert's internal network onto Ernie's internal network.

Firewall rules for Bert
Allow IP packet to flow freely on the loopback device
Deny any packets from the outside world to 127.0.0.0/8 network (loopback address)
These are default rules
100 allow ip from any to any via lo0
200 deny ip from any to 127.0.0.0/8

Drop any netbios traffic traffic on the gif tunnel
300 deny tcp from any to any 135 via gif0
400 deny tcp from any to any 136 via gif0
500 deny tcp from any to any 137 via gif0
600 deny udp from any to any 138 via gif0

Allow isakmp (IKE) between bert & ernie
700 allow udp from 3.3.3.3 500 to 2.2.2.2 500 in via fxp0
800 allow udp from 2.2.2.2 500 to 3.3.3.3 500 out via fxp0

Allow ESP between bert & ernie
900 allow 50 from 3.3.3.3 to 2.2.2.2 in via fxp0
1000 allow 50 from 2.2.2.2 to 3.3.3.3 out via fxp0

Allow ssh to ernie's network and return traffic*
1100 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 22 in via fxp1
1200 allow tcp from 4.4.4.0/24 22 to 1.1.1.0/24 1024-10000 out via fxp1 established

Allow PCAnywhere to ernie's network and return traffic*
1300 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 5631 in via fxp1
1400 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 5632 in via fxp1
1500 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 65301 in via fxp1
1600 allow udp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 5632 in via fxp1
1700 allow tcp from 4.4.4.0/24 5631 to 1.1.1.0/24 1024-10000 out via fxp1 established
1800 allow tcp from 4.4.4.0/24 5632 to 1.1.1.0/24 1024-10000 out via fxp1 established
1900 allow tcp from 4.4.4.0/24 65301 to 1.1.1.0/24 1024-10000 out via fxp1 established
2000 allow udp from 4.4.4.0/24 5632 to 1.1.1.0/24 1024-10000 out via fxp1

http and imap can be handled with dynamic rules
2100 check-state
2200 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 143 keep-state in via fxp1 setup
2300 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 80 keep-state in via fxp1 setup

drop and log everything else
65000 deny log ip from any to any

*Those rules aren't using a state table. They are only looking for the ack or rst bit to be set. Since ernie's network is a trusted network, this is an acceptable risk. As the rule would only allow a stealth scan on high-ports of machines on the 1.1.1.0/24 network.

Firewall rules for Ernie
Allow IP packet to flow freely on the loopback device
Deny any packets from the outside world to 127.0.0.0/8 network (loopback address)
These are default rules
100 allow ip from any to any via lo0
200 deny ip from any to 127.0.0.0/8

Drop any netbios traffic traffic on the gif tunnel
300 deny tcp from any to any 135 via gif0
400 deny tcp from any to any 136 via gif0
500 deny tcp from any to any 137 via gif0
600 deny udp from any to any 138 via gif0

Allow isakmp (IKE) between bert & ernie
700 allow udp from 3.3.3.3 500 to 2.2.2.2 500 out via fxp0
800 allow udp from 2.2.2.2 500 to 3.3.3.3 500 in via fxp0

Allow ESP between bert & ernie

900 allow 50 from 3.3.3.3 to 2.2.2.2 out via fxp0
1000 allow 50 from 2.2.2.2 to 3.3.3.3 in via fxp0

Allow ssh from bert's network*
1100 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 22 out via fxp1
1200 allow tcp from 4.4.4.0/24 22 to 1.1.1.0/24 1024-10000 in via fxp1 established

Allow PCAnywhere from bert's network*
1300 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 5631 out via fxp1
1400 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 5632 out via fxp1
1500 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 65301 out via fxp1
1600 allow udp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 5632 out via fxp1
1700 allow tcp from 4.4.4.0/24 5631 to 1.1.1.0/24 1024-10000 in via fxp1 established
1800 allow tcp from 4.4.4.0/24 5632 to 1.1.1.0/24 1024-10000 in via fxp1 established
1900 allow tcp from 4.4.4.0/24 65301 to 1.1.1.0/24 1024-10000 in via fxp1 established
2000 allow udp from 4.4.4.0/24 5632 to 1.1.1.0/24 1024-10000 in via fxp1

http and imap can be handled with dynamic rules
2100 check-state
2200 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 143 keep-state out via fxp1 setup
2300 allow tcp from 1.1.1.0/24 1024-10000 to 4.4.4.0/24 80 keep-state out via fxp1 setup

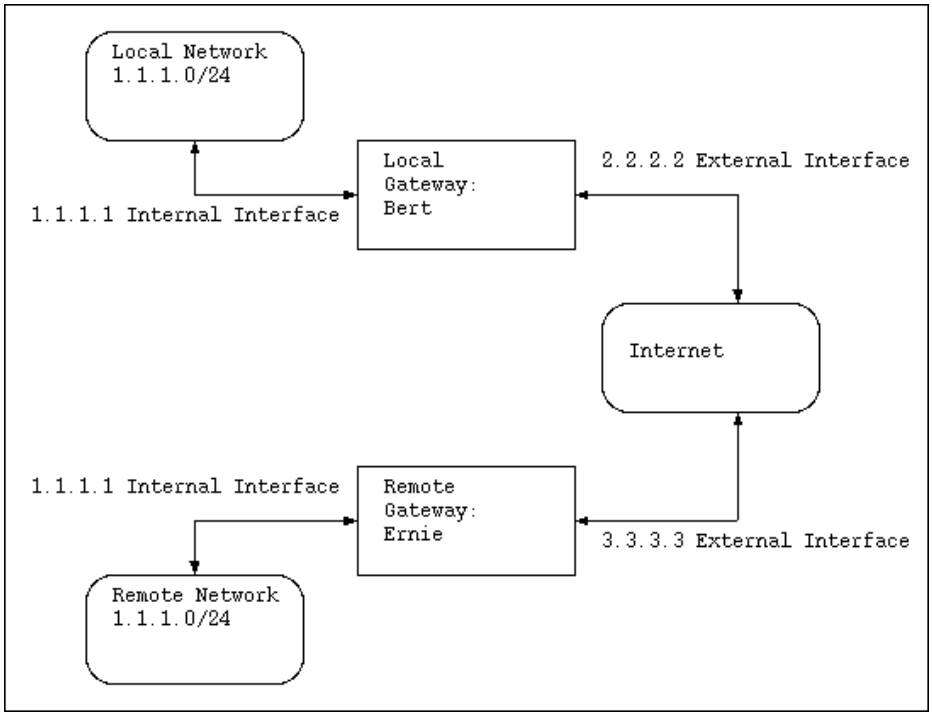
drop and log everything else
65000 deny log ip from any to any

*Same thing as bert's rules. Not using a state table, only looking for the ack or rst bit to be set.

The rules above are just sample rules to give you an idea of how to filter/firewall the ipsec traffic. Using the alias on the loopback device made it easier for me to firewall the ipsec traffic traveling thru the gif tunnel. The ipsec stuff is all handled on the external interface. The private network stuff is handled by the loopback device. This made it easier for me visualize the traffic flow; picture the private network traffic arriving at kernel(between the external & internal interfaces) and then needing to be routed.

NAT

Now that we have covered the basics of setting up an IPSec connection, I'll cover a basic NAT'ing solution. Suppose that both the remote and local networks have the same network number; 1.1.1.0/24. Our network diagram now looks like this:



To keep things simple we'll say the users on Bert's network need to access a single server on Ernie's network.

The goal here will be to allow users on Bert's internal network to access the server 1.1.1.20(Bart) on Ernie's network.

First we setup the IPSec connection like we did above using a gif tunnel and virtual interfaces on the loopback interface. This time we'll use a slightly larger subnet; we'll sue 255.255.255.248 for our subnet. This will allow us to keep our nat aliases in the same subnet as the inside end-points of the gif tunnel. This isn't nessasacry but will help us to remember which connections are using what aliases.

So, on Bert we setup two virtual interfaces: 5.5.5.1 & 5.5.5.4

ifconfig lo0 alias 5.5.5.1 netmask 255.255.255.248
ifconfig lo0 alias 5.5.5.4 netmask 255.255.255.248

On Ernie setup 5.5.5.2 & 5.5.5.3 as the virtual interfaces.

After the virtual interfaces are in place, setup the IPSec connectivity just as before. On Bert you'll need to add a static route 5.5.5.3 and on Ernie you'll need to add a static route for 5.5.5.4 .

On Bert:

route add 5.5.5.3/32 5.5.5.2

On Ernie:

route add 5.5.5.4/32 5.5.5.1

Next we setup NAT using the natd daemon. FreeBSD handbook section on natd can be found here. The man page for natd is here. For nat to work you must have these options in your kernel:

options IPFIREWALL
options IPDIVERT

To provide Bert's users with access to the server 1.1.1.20(Bart) on Ernie's network, Bert's users will use the ip address of 5.5.5.4 as the ip address of the Bart. The Bert and Ernie will do the NAT magic that allows for the access. On Ernie's network Bert's users will all appear to come from the 5.5.5.4 address.

Here is how we set it up:

Going with the same list of interfaces from the Firewall section above.

Firewall rules for Bert to handle NAT'ing. At this point we are working from an empty firewall rule set and only setting up what is needed for NAT.
ipfw add 100 divert natd ip from 1.1.1.0/24 to 5.5.5.3 out via gif0
ipfw add 200 divert natd ip from 5.5.5.3 to 5.5.5.4 in via gif0
ipfw add allow ip from any to any

Next we setup natd on Bert.

natd -alias_address 5.5.5.4

Natd on Bert rewrites the packet so that the source address is 5.5.5.4 and then passes the packet along. When the return traffic arrives natd undoes the change.

Then we setup NAT on Ernie.
First the firewal rules.

ipfw add 100 divert natd ip from 5.5.5.4 to 5.5.5.3 in via gif0
ipfw add 200 divert natd ip from 1.1.1.20 to 5.5.5.4 out via gif0
ipfw add allow ip from any to any

Then setup natd on Ernie.

natd -redirect_address 1.1.1.20 0.0.0.0 -alias_address 5.5.5.3

Natd redirects all traffic to Bart(1.1.1.20). It rewrites the destination address and then passes the packet along. And when the return traffic arrives it undoes the change. Here is the output of natd running on both Bert and Ernie of a ping from a workstation on Bert's network to Bart.

Natd on Bert:

```
Out [ICMP] [ICMP] 1.1.1.60 -> 5.5.5.3 8(0) aliased to
        [ICMP] 5.5.5.4 -> 5.5.5.3 8(0)
In  [ICMP] [ICMP] 5.5.5.3 -> 5.5.5.4 0(0) aliased to
        [ICMP] 5.5.5.3 -> 1.1.1.60 0(0)
Out [ICMP] [ICMP] 1.1.1.60 -> 5.5.5.3 8(0) aliased to
        [ICMP] 5.5.5.4 -> 5.5.5.3 8(0)
In  [ICMP] [ICMP] 5.5.5.3 -> 5.5.5.4 0(0) aliased to
        [ICMP] 5.5.5.3 -> 1.1.1.60 0(0)
```

Natd on Ernie:

```
In  [ICMP] [ICMP] 5.5.5.4 -> 5.5.5.3 8(0) aliased to
        [ICMP] 5.5.5.4 -> 1.1.1.20 8(0)
Out [ICMP] [ICMP] 1.1.1.20 -> 5.5.5.4 0(0) aliased to
        [ICMP] 5.5.5.3 -> 5.5.5.4 0(0)
In  [ICMP] [ICMP] 5.5.5.4 -> 5.5.5.3 8(0) aliased to
        [ICMP] 5.5.5.4 -> 10.1.1.20 8(0)
```

```
Out [ICMP] [ICMP] 10.1.1.20 -> 5.5.5.4 0(0) aliased to
        [ICMP] 5.5.5.3 -> 5.5.5.4 0(0)
```

Saving your work

After it is all working like you want it to, it is best to update the boot-time files. This way a reboot doesn't undo all your hard work.

I'll use bert as reference for this section.

For the ipsec policy, put the setkey commands in /etc/ipsec.conf .
I added the following to /etc/ipsec.conf

```
 spdadd 2.2.2.2 3.3.3.3 any -P out ipsec esp/tunnel/2.2.2.2-3.3.3.3/require;
 spdadd 3.3.3.3 2.2.2.2 any -P in ipsec esp/tunnel/3.3.3.3-2.2.2.2/require;
```

Then edit the /etc/rc.conf file
Add gif0 to the list of network_interface
And add the following lines

```
 ## setup virtual interface for tunnel to ernie
 ifconfig_lo0_alias="inet 5.5.5.1 netmask 255.255.255.252"
 ## load ipsec policy from /etc/ipsec.conf
 ipsec_enable="YES"
 ## configure the ip tunnel to ernie
 gif_interface="gif0"
 gifconfig_gif0="2.2.2.2 3.3.3.3"
 ifconfig_gif0="inet 5.5.5.1 5.5.5.2 netmask 255.255.255.252"
```

Make sure to enable packet forwarding with the following line:

```
  gateway_enable="YES"
```

If you using NAT, then you'll want to also enable that in rc.conf.

```
## Enable natd
 natd_interface="gif0"
 natd_flags="-alias_address 5.5.5.4"
```

Finally you just need to make sure racoon starts up. I added the following racoon.sh script to /usr/local/etc/rc.d :

```
----------------------------------------->8-------------------------------------------------------------------------

 #!/bin/sh
 if ! PREFIX=$(expr $0 : "\(/.*\)/etc/rc\.d/$(basename $0)\$"); then
 echo "$0: Cannot determine the PREFIX" >&2
 exit 1
 fi
 case "$1" in
 start)
 [ -x ${PREFIX}/sbin/racoon ] && ${PREFIX}/sbin/racoon && echo -n ' racoon'
 ;;
 stop)
 killall racoon && echo -n ' racoon stopped'
 ;;
 *)
 echo "Usage: `basename $0` {start|stop}" >&2
 ;;
 esac
 exit 0

----------------------------------------->8-------------------------------------------------------------------------
```

Make sure it is executable; chmod +x /usr/local/etc/rc.d/racoon.sh

And since I included sample firewall rules, I should show a way of having those rules initiated at boot.
In /etc/rc.conf add the following lines

```
firewall_enable="YES"
firewall_script="/etc/rc.firewall"
firewall_type="bert"
```

Then in /etc/rc.firewall between ";;" and "[Uu][Nn][Kk][Nn][Oo][Ww][Nn])" stick in your firewall rules
Listed below are the lines to add for bert's firewall rules from firewall section above.

```
[Bb][Ee][Rr][Tt])
# setup some variables
# oif is the external interface
# oip is the external IP address
# iif is the internal interface
# iip is the internal IP address
```

```
# inet is the internal network
# rnet is the remote network
# rmip is the remote gateway's external IP address
# op is the outbound port number range

oif="fxp0"
oip="2.2.2.2"
iif="fxp1"
iip="1.1.1.1"
inet="1.1.1.0/24"
rnet="4.4.4.0/24"
rmip="3.3.3.3"
op="1024-10000"

# Drop any netbios traffic traffic on the gif tunnel
${fwcmd} add deny tcp from any to any 135 via gif0
${fwcmd} add deny tcp from any to any 136 via gif0
${fwcmd} add deny tcp from any to any 137 via gif0
${fwcmd} add deny udp from any to any 138 via gif0

# Allow isakmp (IKE) between bert & ernie
${fwcmd} add allow udp from ${rmip} 500 to ${oip} 500 in via ${oif}
${fwcmd} add allow udp from ${oip} 500 to ${rmip} 500 out via ${oif}

# Allow ESP between bert & ernie
${fwcmd} add allow 50 from ${rmip} to ${oip} in via ${oif}
${fwcmd} add allow 50 from ${oip} to ${rmip} out via ${oif}

# Allow ssh to ernie's network and return traffic
# not using a state table here - you have been warned :-)
${fwcmd} add allow tcp from ${inet} ${op} to ${rnet} 22 in via ${iif}
${fwcmd} add allow tcp from ${rnet} 22 to ${inet} ${op} out via ${iif} established

# Allow PCAnywhere to ernie's network and return traffic
# not using a state table here - you have been warned :-)
${fwcmd} add allow tcp from ${inet} ${op} to ${rnet} 5631 in via ${iif}
${fwcmd} add allow tcp from ${inet} ${op} to ${rnet} 5632 in via ${iif}
${fwcmd} add allow tcp from ${inet} ${op} to ${rnet} 65301 in via ${iif}
${fwcmd} add allow udp from ${inet} ${op} to ${rnet} 5632 in via ${iif}
${fwcmd} add allow tcp from ${rnet} 5631 to ${inet} ${op} out via ${iif} established
${fwcmd} add allow tcp from ${rnet} 5632 to ${inet} ${op} out via ${iif} established
${fwcmd} add allow tcp from ${rnet} 65301 to ${inet} ${op} out via ${iif} established
${fwcmd} add allow udp from ${rnet} 5632 to ${inet} ${op} out via ${iif}

# http and imap can be handled with dynamic rules
# here we are using a state table
${fwcmd} add check-state
${fwcmd} add allow tcp from ${inet} ${op} to ${rnet} 143 keep-state in via ${iif} setup
${fwcmd} add allow tcp from ${inet} ${op} to ${rnet} 80 keep-state in via ${iif} setup

# drop and log everything else
${fwcmd} add 65000 deny log ip from any to any
```

It would probably also be wise to reboot the device after making these changes to confirm everything works properly after a reboot.

Summary

Hopefully the above information will provide you with another viewpoint on setting up IPSec connections. As IPv6 slowly takes over, the need to NAT and possbility of network number conflicts is reduced. But until then the information provided here should be useful for connecting WANs across public networks and providing a solution to network numbering conflicts.

Please remember the firewall rules and nat configuration are provided as examples only of what is possible. You should evaluate your needs before putting firewall rules and/or nat rules in place.

References and Resources

My previous notes on IPSec
Useful IPSec how-to
Another example of setting up an IPSec tunnel
rfc2041
rfc2406
Kame website - authors of Racoon
IPSec section of the FreeBSD Handbook
IPFW section of the FreeBSD Handbook
NAT section of the FreeBSD Handbook
Man page for gif
Man page for gifconfig
Man page for ipfw

Man page for natd
Man page for setkey