IPsec HOWTO

Ralf Spenneberg

ralf (at) spenneberg.net

2003-08-18

Revision History

Revision 0.9.96	2007-02-26	Revised by: RS
OpenSSL needs file: crlnumber		
Revision 0.9.95	2005-09-03	Revised by: RS
Added iptables rule setting the MSS and one minor correction		
Revision 0.9.94	2005-07-19	Revised by: RS
Added some remarks about routing		
Revision 0.9.93	2005-03-3	Revised by: RS
fwd-policy corrected, p12 added		
Revision 0.9.92	2005-02-1	Revised by: RS
fwd-policy added		
Revision 0.9.91	2005-01-31	Revised by: RS
/etc/ipsec.conf replaced by /etc/setkey.conf		
Revision 0.9.9	2004-12-22	Revised by: RS
Nat-Traversal added Changed Document structure		
Revision 0.9.6	2004-01-28	Revised by: RS
Correction modp768		
Revision 0.9.5	2004-01-08	Revised by: RS
Added Compilation of certpatch and keyconv		
Revision 0.9.4	2003-08-28	Revised by: RS
Corrections		
Revision 0.9.3	2003-08-22	Revised by: RS
Fixed a typo		
Revision 0.9.2	2003-08-19	Revised by: RS
Fixed a typo		
Revision 0.9.1	2003-08-18	Revised by: RS
Minor corrections		
Revision 0.9.0	2003-08-15	Revised by: RS
Added: Using the OpenBSD isakmpd		
Revision 0.8.3	2003-05-13	Revised by: RS
Further typos corrected. Some sentences rephrased.		
Revision 0.8.2	2003-05-03	Revised by: RS
Bugfixes		
Revision 0.8.1	2003-04-30	Revised by: RS
added chapter covering certificates		
Revision 0.8	2003-04-18	Revised by: RS
first draft		

This HowTo will cover the basic and advanced steps setting up a VPN using IPsec based on the Linux Kernels 2.6. Since there is a vast amount of documentation available for the Linux Kernel 2.4, this HowTo will concentrate on the new IPsec Features in the 2.6 kernel.

Table of Contents

Introduction 3
Theory 4
Openswan running on Linux 2.6 9
Linux Kernel 2.6 using KAME-tools 9
Linux Kernel 2.6 using OpenBSD's isakmpd 20
Generating X.509 Certificates 24
Advanced Configuration 28
Links 28

Introduction

The latest version of this document can always be found at The Linux Documentation Project[1] and at the official homepage http://www.ipsec-howto.org.

Reasons to write this HowTo

I have used numeruos HowTos in the past. Most were very valuable to me. When the new IPsec features in the Linux Kernel were implemented I started to play around using them. Soon I found out that only very little documentation exists. That started me writing this HowTo.

Format of this document

This document is broken down into 7 chapters.

Section 1: Introduction

This section

Section 2: Theory

IPsec theory. Essentially the IPsec protocols.

Section 3: Openswan

This section will describe how to setup Openswan on the Kernel 2.6.

Section 4: Racoon running on Linux Kernel 2.6

This section describes how to setup an IPsec VPN using the KAME tools **setkey** and **racoon**. This now includes NAT-Traversal.

Section 5: Isakmpd running on Linux Kernel 2.6

This section describes how to setup an IPsec VPN using OpenBSD isakmpd IKE daemon.

Section 6: Generating X.509 Certificates

This section describes how to generate X.509 Certificates using the **openssl**-Command.

Section 7: Advanced Configuration

This section gives some hints on XAUTH and on useful **iptables**-rules.

Contributors to this document

- Matija Nalis
- Fridtjof Busse
- Uwe Beck
- Juanjo Ciarlante
- Ervin Hegedus
- Barabara Kane
- Alois Schmid

Legal Information

Copyright

Disclaimer

The author assumes no responsibility for anything done with this document, nor does he make any warranty, implied or explicit. If your dog dies, the author may not be made responsible!

Related Documents

- Networking Overview HOWTO[3]
- Networking HOWTO[4]
- VPN-Masquerade HOWTO[5]
- VPN HOWTO[6]
- Advanced Routing & Traffic Control HOWTO[7]

Theory

What is IPsec?

IPsec is an extension to the IP protocol which provides security to the IP and the upper-layer protocols. It was first developed for the new IPv6 standard and then "backported" to IPv4. The IPsec architecture is described in the RFC2401. The following few paragraphs will give you a short introduction into IPsec.

IPsec uses two different protocols - AH and ESP - to ensure the authentication, integrity and confidentiality of the communication. It can protect either the entire IP datagram or only the upper-layer protocols. The appropiate modes are called tunnel mode and transport mode. In tunnel mode the IP datagram is fully encapsulated by a new IP datagram using the IPsec protocol. In transport mode only the payload of the IP datagram is handled by the IPsec protocol inserting the IPsec header between the IP header and the upper-layer protocol header (see Figure 1).

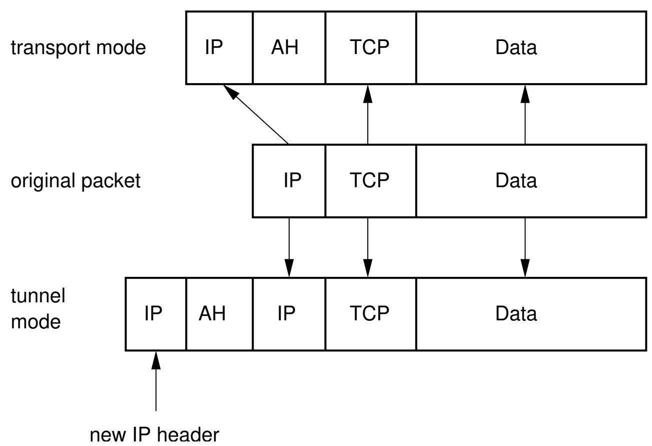

Figure 1. IPsec tunnel and transport mode

To protect the integrity of the IP datagrams the IPsec protocols use hash message authentication codes (HMAC). To derive this HMAC the IPsec protocols use hash algorithms like MD5 and SHA to calculate a hash based on a secret key and the contents of the IP datagram. This HMAC is then included in the IPsec protocol header and the receiver of the packet can check the HMAC if it has access to the secret key.

To protect the confidentiality of the IP datagrams the IPsec protocols use standard symmetric encryption algorithms. The IPsec standard requires the implementation of NULL and DES. Today usually stronger algorithms are used like 3DES, AES and Blowfish.

To protect against denial of service attacks the IPsec protocols use a sliding window. Each packet gets assigned a sequence number and is only accepted if the packet's number is within the window or newer. Older packets are immediately discarded. This protects against replay attacks where the attacker records the original packets and replays them later.

For the peers to be able to encapsulate and decapsulate the IPsec packets they need a way to store the secret keys, algorithms and IP addresses involved in the communication. All these parameters needed for the protection of the IP datagrams are stored in a security association (SA). The security associations are in turn stored in a security association database (SAD).

Each security association defines the following parameters:

- Source and destination IP address of the resulting IPsec header. These are the IP addresses of the IPsec peers protecting the packets.
- IPsec protocol (AH or ESP), sometimes compression (IPCOMP) is supported, too.
- The algorithm and secret key used by the IPsec protocol.
- Security Parameter Index (SPI). This is a 32 bit number which identifies the security association.

Some implementations of the security association database allow further parameters to be stored:

- IPsec mode (tunnel or transport)
- Size of the sliding window to protect against replay attacks.
- Lifetime of the security association.

Since the security association defines the source and destination IP addresses, it can only protect one direction of the traffic in a full duplex IPsec communication. To protect both directions IPsec requires two unidirectional security associations.

The security assocations only specify how IPsec is supposed to protect the traffic. Additional information is needed to define which traffic to protect when. This information is stored in the security policy (SP) which in turn is stored in the security policy database (SPD).

A security policy usually specifies the following parameters:

- Source and destination address of the packets to be protected. In transport mode these are the same addresses as in the SA. In tunnel mode theymay differ!
- The protocol (and port) to protect. Some IPsec implementations do not allow the definition of specific protocols to protect. In this case all traffic between the mentioned IP addresses is protected.
- The security association to use for the protection of the packets.

The manual setup of the security association is quite error prone and not very secure. The secret keys and encryption algorithms must be shared between all peers in the virtual private network. Especially the exchange of the keys poses critical problems for the system administrator: How to exchange secret symmetric keys when no encryption is yet in place?

To solve this problem the internet key exchange protocol (IKE) was developed. This protocol authenticates the peers in the first phase. In the second phase the security associations are negotiated and the secret symmetric keys are chosen using a Diffie Hellmann key exchange. The IKE protocol then even takes care of periodically rekeying the secret keys to ensure their confidentiality.

IPsec Protocols

The IPsec protocol family consists of two protocols: Authentication Header (AH) and Encapsulated Security Payload (ESP). Both are independent IP protocols. AH is the IP protocol 51 and ESP is the IP protocol 50 (see `/etc/protocols`). The following two sections will briefly cover their properties.

AH - Authentication Header

The AH protocol protects the integrity of the IP datagram. To achieve this, the AH protocol calculates a HMAC to protect the integrity. When calculating the HMAC the AH protocol bases it on the secret key, the payload of the packet and the immutable parts of the IP header like the IP addresses. It then adds the AH header to the packet. The AH header is shown in Figure 2.

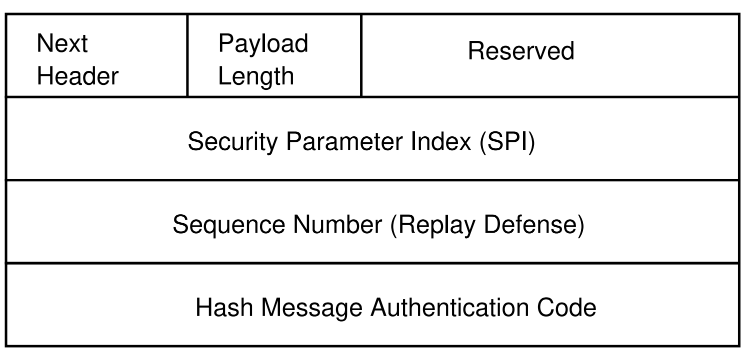

Figure 2. The AH Header protect the integrity of the packet

The AH header is 24 bytes long. The first byte is the *Next Header* field. This field specifies the protocol of the following header. In tunnel mode a complete IP datagram is encapsulated; therefore the value of this field is 4. When encapsulating a TCP datagram in transport mode the corresponding value is 6. The next byte specifies the length of the payload. This field is followed by two reserved bytes. The next double word specifies the 32 bit long *Security Parameter Index* (SPI). The SPI specifies the security association to use for the decapsulation of the packet. The 32 bit *Sequence Number* protects against replay attacks. Finally the 96 bit holds the *hash message authentication code* (HMAC). This HMAC protects the integrity of the packets since only the peers knowing the secret key can create and check the HMAC.

Since the AH protocol protects the IP datagram including immutable parts of the IP header like the IP addresses the AH protocol does not allow NAT. Network address translation (NAT) replaces an IP address in the IP header (usually the source IP) by a different IP address. After the exchange the HMAC is not valid anymore. The NAT-Traversal extension of the IPsec protocol implements ways around this restriction.

ESP - Encapsulated Security Payload

The ESP protocol can both ensure the integrity of the packet using a HMAC and the confidentiality using encryption. After encrypting the packet and calculating the HMAC the ESP header is generated and added to the packet. The ESP header consists of two parts and is shown in Figure 3.

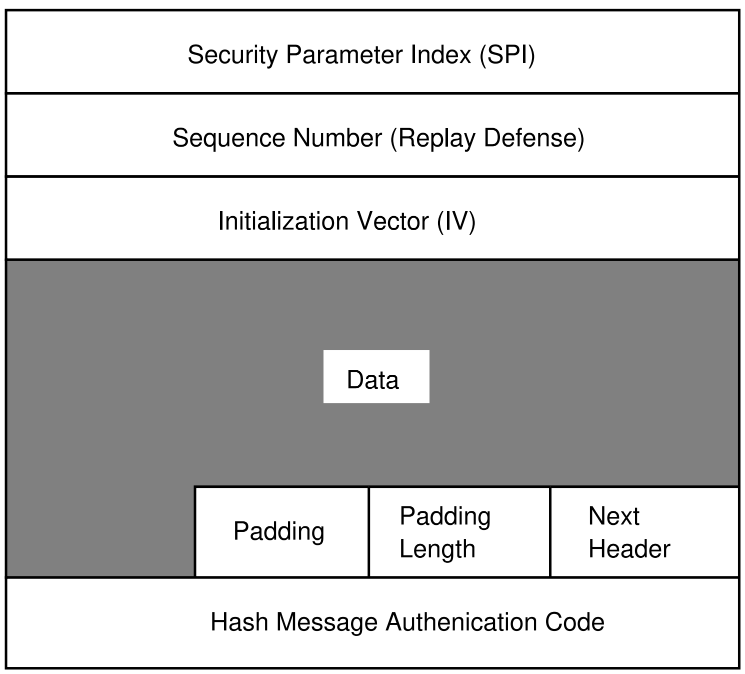

Figure 3. The ESP header

The first doubleword in the ESP header specifies the *Security Parameter Index* (SPI). This SPI specifies the SA to use for the decapsulation of the ESP packet. The second doubleword holds the *Sequence Number*. This sequence number is used to protect against replay attacks. The third doubleword specifies the *Initialization Vector* (IV) which is used in the encryption process. Symmetric encryption algorithms are susceptible to a frequency attack if no IV is used. The IV ensures that two identical payloads lead to different encrypted payloads.

IPsec uses block ciphers for the encryption process. Therefore the payload may need to be padded if the length of the payload is not a multiple of the block length. The length of the pad is then added. Following the pad length the 2 byte long *Next Header* field specifies the next header. Lastly the 96 bit long HMAC is added to the ESP header ensuring the integrity of the packet. This HMAC only takes the payload of the packet into account. The IP header is not include in the calculation process.

The usage of NAT therefore does not break the ESP protocol. Still in most cases NAT is not possible in combination with IPsec. The NAT-Traversal offers a solution in this case by encapsulating the ESP packets within UDP packets.

IKE Protocol

The IKE protocol solves the most prominent problem in the setup of secure communication: the authentication of the peers and the exchange of the symmetric keys. It then creates the security associations and populates the SAD. The IKE protocol usually requires a user space daemon and is not implemented in the operating system. The IKE protocol uses 500/udp for it's communication.

The IKE protocol functions in two phases. The first phase establishes a *Internet Security Association Key Management Security Association* (ISAKMP SA). In the second phase the ISAKMP SA is used to negotiate and setup the IPsec SAs.

The authentication of the peers in the first phase can usually be based on pre-shared keys (PSK), RSA keys and X.509 certificates (**racoon** even supports Kerberos).

The first phase usually supports two different modes: main mode and aggressive mode. Both modes authenticate the peer and setup an ISAKMP SA, but the aggressive mode uses only half the number of messages to achieve this goal. This does have its drawbacks though, because the aggressive mode does not support identity protection and is therefore susceptible to a man-in-the-middle attack if used in conjunction with pre-shared keys. On the other hand this is the only purpose of the aggressive mode. Because of the internal workings of the main mode it does not support the usage of different preshared keys with unknown peers. The aggressive mode does not support identity protection and transfers the identity of the client in the clear. The peers therefore know each other before the authentication takes place and different pre-shared keys can be used for different peers.

In the second phase the IKE protocol exchanges security association proposals and negotiates the security associations based on the ISAKMP SA. The ISAKMP SA provides the authentication to protect against a man-in-the-middle attack. This second phase uses the quick mode.

Usually two peers negotiate only one ISAKMP SA, which is then used to negotiate several (at least two) unidirectional IPsec SAs.

NAT-Traversal

What is NAT-Traversal and why is it needed?

Often one peer in the VPN is behind a NAT-device. I just assume Source-NAT devices here. Whenever I talk about NAT I mean Source-NAT or Masquerading. What does this mean concerning the VPN? Well, first of all the original IP address of the peer is hidden by the NAT-device. The NAT-device conceals the original source IP address and replaces it by its own IP address.

This make the IPsec AH protocol immediately unusable. But ESP can still be used if both sides are configured correctly.

So why do you need NAT-Traversal? Because as soon as two machines behind the same NAT device try to build a tunnel to the outside, both will fail.

Why is this happening? The NAT device needs to keep track of the "natted" connections to be able to "de-nat" the reply packets back to the original client. Therefore the NAT device maintains an internal table where all "natted" connections are stored. Lets assume one client connects to a webserver on the Internet. The NAT device conceils the original address by replacing it with its own address. It then makes a note in its internal table that all packets coming back on the chosen client port have to be send to the original client1. As soon as the second client starts a connection, it handles that connection identical. If the second client chose the same client port by coincidence the NAT device will also modify the client port for unambuigity. This works very well using TCP and UDP because those protocols provide ports. ESP does not use ports. Therefore the NAT device can only use the protocol distinguish the packets. When the first client connects it stores the information in the table that all ESP packets have to be "denatted" to the first client. When the second client connects it will overwrite this entry with the appropiate entry for the second one thus breaking at least the first connection.

What does NAT traversal do to help? NAT-traversal again encapsulates the ESP packets in UDP packets. These can easily be handled by a NAT device since they provide ports. By default port 4500/udp is used. NAT traversal is specified in several drafts. There are no RFCs at the moment. A nice feature of NAT traversal is the fact that once activated the peers automatically use it when needed.

Openswan running on Linux 2.6

ToDo

Linux Kernel 2.6 using KAME-tools

This chapter explains the usage of the native IPsec stack of the Linux Kernel $\geq$2.5.47 and 2.6.*. The installation and the configuration of this IPsec stack differs greatly from FreeS/WAN and is similar to the *BSD variants like FreeBSD, NetBSD and OpenBSD.

I will first cover the configuration and installation of the Linux kernel and the user space tools. Then the setup of a manually keyed connection in transport and tunnel mode will be explained. Finally we will cover the setup of automatically keyed connections using preshared keys and X.509 certificates. The support of roadwarriors will be explained last.

Installation

The installation requires at least a Linux kernel of version 2.5.47 or 2.6.*. The kernel source may be downloaded at http://www.kernel.org. After downloading the source the kernel source package must be extracted, configured and compiled.

```
cd /usr/local/src
tar xvjf /path-to-source/linux-<version>.tar.bz2
cd linux-<version>
make xconfig
make bzImage
make modules
make modules_install
make install
```

These are the most often used commands to configure and compile the Linux kernel. If you need a special setup please refer to the Kernel-Howto.

When configuring the kernel, it is important, to turn on the following features:

```
Networking support (NET) [Y/n/?] y
  *
  * Networking options
  *
  PF_KEY sockets (NET_KEY) [Y/n/m/?] y
  IP: AH transformation (INET_AH) [Y/n/m/?] y
  IP: ESP transformation (INET_ESP) [Y/n/m/?] y
  IP: IPsec user configuration interface (XFRM_USER) [Y/n/m/?] y

Cryptographic API (CRYPTO) [Y/n/?] y
  HMAC support (CRYPTO_HMAC) [Y/n/?] y
  Null algorithms (CRYPTO_NULL) [Y/n/m/?] y
  MD5 digest algorithm (CRYPTO_MD5) [Y/n/m/?] y
  SHA1 digest algorithm (CRYPTO_SHA1) [Y/n/m/?] y
  DES and Triple DES EDE cipher algorithms (CRYPTO_DES) [Y/n/m/?] y
  AES cipher algorithms (CRYPTO_AES) [Y/n/m/?] y
```

Depending on the version of the kernel used you might have to turn on IPv6 support too.

Once the kernel is compiled and installed the user space tools may be installed. Currently the tools are maintained at http://ipsec-tools.sourceforge.net/[9]. When compiling the package by hand you may need to specify the location of the kernel headers. This package needs the kernel headers of at least kernel version 2.5.47.

Attention: When using Linux kernel >= 2.6.10 you must use the ipsec-tools >=0.5 because this kernel added a new forward policy unknown to racoon in the older ipsec-tools. Be aware that some Linux distributions heavily patch even older Linux kernels so this may apply to you, too. Just check for **fwd** policies in your kernel.

```
./configure --with-kernel-headers=/lib/modules/2.6.X/build/include
make
make install
```

Now everything should be ready to go.

Manual keyed connections using setkey

A manual keyed connection means that all parameters needed for the setup of the connection are provided by the administrator. The IKE protocol is not used to automatically authenticate the peers and negotiate these parameters. The administrator decides which protocol, algorithm and key to use for the creation of the security associations and populates the security association database (SAD) accordingly.

Transport Mode

This section will first cover the setup of a manual keyed connection in transport mode. This is probably the best way to start because it is the simplest connection to setup. This section assumes that two machines with the IP addresses 192.168.1.100 and 192.168.2.100 communicate using IPsec.

All parameters stored in the SAD and the SPD can be modified using the **setkey** command. This command has a quite exhaustive man page. Therefore only the options needed for the setup of a connection in transport mode are covered here. **setkey** reads its commands from a file when invoked with **setkey -f /etc/setkey.conf**. A suitable `/etc/setkey.conf` file is shown in following listing.

```
#!/usr/sbin/setkey -f

# Configuration for 192.168.1.100
```

```
# Flush the SAD and SPD
flush;
spdflush;

# Attention: Use this keys only for testing purposes!
# Generate your own keys!

# AH SAs using 128 bit long keys
add 192.168.1.100 192.168.2.100 ah 0x200 -A hmac-md5
0xc0291ff014dccdd03874d9e8e4cdf3e6;
add 192.168.2.100 192.168.1.100 ah 0x300 -A hmac-md5
0x96358c90783bbfa3d7b196ceabe0536b;

# ESP SAs using 192 bit long keys (168 + 24 parity)
add 192.168.1.100 192.168.2.100 esp 0x201 -E 3des-cbc
0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;
add 192.168.2.100 192.168.1.100 esp 0x301 -E 3des-cbc
0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df;

# Security policies
spdadd 192.168.1.100 192.168.2.100 any -P out ipsec
           esp/transport//require
           ah/transport//require;

spdadd 192.168.2.100 192.168.1.100 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

You will need some keys to replace the keys of this script, if you want to use the manually keyed connection for anything but testing purposes. Use a command such as the following to generate your keys:

```
$ # 128 Bit long key
$ dd if=/dev/random count=16 bs=1| xxd -ps
16+0 Records ein
16+0 Records aus
cd0456eff95c5529ea9e918043e19cbe

$ # 192 Bit long key
$ dd if=/dev/random count=24 bs=1| xxd -ps
24+0 Records ein
24+0 Records aus
9d6c4a8275ab12fbfdcaf01f0ba9dcfb5f424c878e97f888
```

Please use the device **/dev/random** when generating the keys because it ensures random keys.

The script first flushes the security association database (SAD) and the security policy database (SPD). It then creates AH SAs and ESP SAs. The command **add** adds a security association to the SAD and requires the source and destination IP address, the IPsec protocol (**ah**), the SPI (**0x200**) and the algorithm. The authentication algorithm is specified with **-A** (encryption using **-E**, compression using **-C**; IP compression is not yet supported). Following the algorithm the key must be specified. The key may be formatted in double-quoted "ASCII" or in hexadecimal with a leading **0x**.

Linux supports the following algorithms for AH and ESP: hmac-md5 and hmac-sha, des-cbc and 3des-cbc. Within a short amount of time the following algorithms will probably be supported: simple (no encryption), blowfish-cbc, aes-cbc, hmac-sha2-256 and hmac-sha2-512.

spdadd adds the security policies to the SPD. These policies define which packets are to be protected by IPsec and which protocols and keys to use. The command requires the source and destination IP addresses of the packets to be protected, the protocol (and port) to protect (any) and the policy to use (-P). The policy specifies the direction (in/out), the action to apply (ipsec/discard/none), the protocol (ah/esp/ipcomp), the mode (transport) and the level (use/require).

This configuration file has to be created on both peers taking part in the IPsec communication. While the shown listing works without any change on the peer 192.168.1.100 it has to be slightly modified on the peer 192.168.2.100 to reflect the change of direction of the packets. The easiest way to do it is to exchange the directions in the security policies: replace **-P in** with **-P out** and vice versa. This is shown in the following listing:

```
#!/usr/sbin/setkey -f

# Configuration for 192.168.2.100

# Flush the SAD and SPD
flush;
spdflush;

# Attention: Use this keys only for testing purposes!
# Generate your own keys!

# AH SAs using 128 bit long keys
add 192.168.1.100 192.168.2.100 ah 0x200 -A hmac-md5
0xc0291ff014dccdd03874d9e8e4cdf3e6;
add 192.168.2.100 192.168.1.100 ah 0x300 -A hmac-md5
0x96358c90783bbfa3d7b196ceabe0536b;

# ESP SAs using 192 bit long keys (168 + 24 parity)
add 192.168.1.100 192.168.2.100 esp 0x201 -E 3des-cbc
0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831;
add 192.168.2.100 192.168.1.100 esp 0x301 -E 3des-cbc
0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df;

# Security policies
spdadd 192.168.1.100 192.168.2.100 any -P in ipsec
           esp/transport//require
           ah/transport//require;

spdadd 192.168.2.100 192.168.1.100 any -P out ipsec
           esp/transport//require
           ah/transport//require;
```

Once the configuration file is in place on the peers it can be loaded using **setkey -f /etc/setkey.conf**. The successful load can be tested by displaying the SAD and the SPD:

```
# setkey -D
# setkey -DP
```

The setup resembles now the setup of Figure 4.

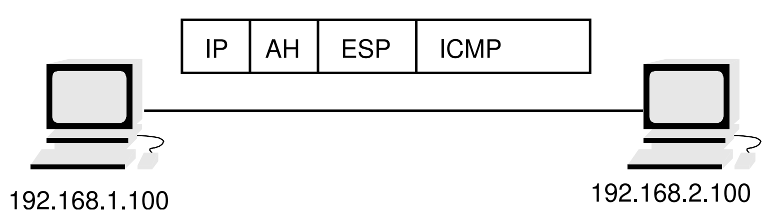

Figure 4. Two machines in transport mode using AH and ESP

If you now ping from one peer to the other the traffic will be encrypted and tcpdump will show the following packets:

```
12:45:39.373005 192.168.1.100 > 192.168.2.100: AH(spi=0x00000200,seq=0x1):
ESP(spi=0x00000201,seq=0x1) (DF)
12:45:39.448636 192.168.2.100 > 192.168.1.100: AH(spi=0x00000300,seq=0x1):
ESP(spi=0x00000301,seq=0x1)
12:45:40.542430 192.168.1.100 > 192.168.2.100: AH(spi=0x00000200,seq=0x2):
ESP(spi=0x00000201,seq=0x2) (DF)
12:45:40.569414 192.168.2.100 > 192.168.1.100: AH(spi=0x00000300,seq=0x2):
ESP(spi=0x00000301,seq=0x2)
```

Tunnel Mode

Tunnel mode is used when the two peers using IPsec work as a gateway and protect the traffic between two networks (Figure 5). The original IP packets are encrypted and encapsulated by one gateway and transfered to it's peer. The peer will decapsulate the packet and will pass on the original unprotected packet.

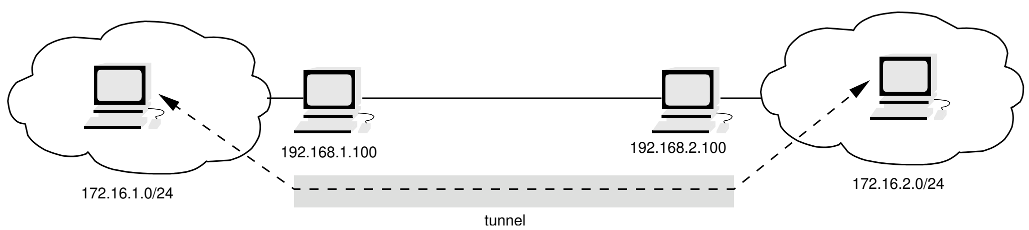

Figure 5. The two peers protect the traffic between two networks

The configuration of the security associations and policies for the tunnel mode is similar to the transport mode and is shown in the following listing.

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

# ESP SAs doing encryption using 192 bit long keys (168 + 24 parity)
# and authentication using 128 bit long keys
add 192.168.1.100 192.168.2.100 esp 0x201 -m tunnel -E 3des-cbc
0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
-A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 192.168.2.100 192.168.1.100 esp 0x301 -m tunnel -E 3des-cbc
0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
-A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;

# Security policies
spdadd 172.16.1.0/24 172.16.2.0/24 any -P out ipsec
           esp/tunnel/192.168.1.100-192.168.2.100/require;

spdadd 172.16.2.0/24 172.16.1.0/24 any -P in ipsec
           esp/tunnel/192.168.2.100-192.168.1.100/require;
```

Attention: When using the Linux kernel >= 2.6.10 you also have to define the forward policy if packets need to be fowarded by the box. Just make sure you use the ipsec-tools 0.5 which add this policy automatically or add it yourself if using older tools. If you are running setkey in Kernel-mode (-k) you have to add the fwd-policy manually, too.

```
spdadd 172.16.2.0/24 172.16.1.0/24 any -P fwd ipsec
           esp/tunnel/192.168.2.100-192.168.1.100/require;
```

This example uses only the ESP protocol. The ESP protocol can ensure integrity and confidentiality. In this case the order of the ESP algorithms is important. First you need to define the encryption algorithm and its key and secondly the authentication algorithm and its key.

For the peer of the tunnel you have to copy this file and to replace the direction of the policies (**in** vs. **out**). If you are using a forward policy, you have to additionally reverse the directions of the IP addresses.

In contrast to the BSD IPsec implementation a security association on Linux can only be used for either transport or tunnel mode. Transport mode is the default mode, so whenever tunnel mode is desired, the security association has to be defined with **-m tunnel**.

The security policies now specify the IP addresses of the protected networks. Packets using these source and destination IP addresses shall be protected by IPsec. Whenever the tunnel mode is used the security policy must specify tunnel and the IP addresses of the actual peers doing implementing the protection. This information is needed to find the appropiate IPsec SA.

If you tunnel is not working, please check your routing. Your hosts need to know that they should send the packets for the opposite network to you vpn gateway. The easiest setup would be using your vpn gateway as default gateway.

Automatic keyed connections using racoon

The KAME IKE daemon **racoon** has also been ported to Linux. This daemon is able to setup automatically keyed IPsec connections. Racoon supports the authentication using preshared keys, X.509 certificates and Kerberos. The daemon can use main mode, aggressive mode and base mode in phase one of IKE. This chapter will show the configuration of **racoon** in main mode using preshared keys and X.509 certificates (ToDo: Kerberos). At the end the configuration of a roadwarrior scenario will be briefly explained.

Remember: If you are using the Linux kernel 2.6.10 (or a heavily patched 2.6.9 by your distribution) you need the ipsec-tools 0.5.

Preshared Keys

The easiest way to authenticate using **racoon** is the usage of preshared keys. These keys have to be defined in a file `/etc/psk.txt`. This file should not be read by unprivileged users (**chmod 400 /etc/psk.txt**) and has the following syntax:

```
# IPv4 Adressen
192.168.2.100          simple psk
5.0.0.1                0xe10bd52b0529b54aac97db63462850f3
# USER_FQDN
ralf@spenneberg.net    This is a psk for an email address
# FQDN
www.spenneberg.net     This is a psk
```

This file is organized in columns. The first column holds the identity of the peer authenticated by the psk. Everything starting in the second column is the PSK.

The configuration of **racoon** is straightforward. The following listing shows a typical `/etc/racoon.conf` configuration file:

```
path pre_shared_key "/etc/psk.txt";

remote 192.168.2.100 {
        exchange_mode main;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;
        }
}

sainfo address 172.16.1.0/24 any address 172.16.2.0/24 any {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

This configuration file first defines where **racoon** may find the preshared keys. It then defines a peer 192.168.2.100 and the parameters to use for the phase one of the IKE negotiation. The second paragraph specifies the parameters which may be used for the setup of the security associations. This definition may be specific for defined IP addresses or general using **anonymous** instead of the IP addresses. Here the encryption, authentication and compression algorithms to use for the SA are defined. All three need to be defined to avoid an error during the startup of **racoon**.

The IKE daemon **racoon** does not start the tunnel negotiation immediately when started. Rather **racoon** waits until the tunnel is needed. For this notification to occur the kernel needs to know when to notify **racoon**. To achieve this, the administrator needs to define security policies without the appropiate security associations. Whenever the Linux kernel needs to protect a packet according to the security policies and when no security association is available, the Linux kernel calls **racoon** and asks for the required security associations. **Racoon** will then start the IKE negotiations and will create the SAs when finished. The Linux kernel can then send the packets.

For the assumed setup the following policies are needed on 192.168.1.100:

```
#!/usr/sbin/setkey -f
#
# Flush SAD and SPD
flush;
spdflush;

# Create policies for racoon
spdadd 172.16.1.0/24 172.16.2.0/24 any -P out ipsec
           esp/tunnel/192.168.1.100-192.168.2.100/require;

spdadd 172.16.2.0/24 172.16.1.0/24 any -P in ipsec
           esp/tunnel/192.168.2.100-192.168.1.100/require;
```

Once the policies are loaded using **setkey -f /etc/setkey.conf racoon** may be started. For testing purposes **racoon** should be started using **racoon -F -f /etc/racoon.conf**. Again the configuration of the other peer has to be modified to reflect the different direction. The IP addresses in the files `/etc/psk.txt`, `/etc/setkey.conf` and `/etc/racoon.conf` must be exchanged.

The initiation of the tunnel can then be followed in the logs:

```
2003-02-21 18:11:17: INFO: main.c:170:main(): @(#)racoon 20001216 20001216
```

```
 sakane@kame.net
2003-02-21 18:11:17: INFO: main.c:171:main(): @(#)This product linked Open
SSL 0.9.6b [engine] 9 Jul 2001 (http://www.openssl.org/)
2003-02-21 18:11:17: INFO: isakmp.c:1365:isakmp_open(): 127.0.0.1[500] use
d as isakmp port (fd=7)
2003-02-21 18:11:17: INFO: isakmp.c:1365:isakmp_open(): 192.168.1.100[500]
 used as isakmp port (fd=9)
2003-02-21 18:11:37: INFO: isakmp.c:1689:isakmp_post_acquire(): IPsec-SA r
equest for 192.168.2.100 queued due to no phase1 found.
2003-02-21 18:11:37: INFO: isakmp.c:794:isakmp_ph1begin_i(): initiate new
phase 1 negotiation: 192.168.1.100[500]<=>192.168.2.100[500]
2003-02-21 18:11:37: INFO: isakmp.c:799:isakmp_ph1begin_i(): begin Identit
y Protection mode.
2003-02-21 18:11:37: INFO: vendorid.c:128:check_vendorid(): received Vendor
 ID: KAME/racoon
2003-02-21 18:11:37: INFO: vendorid.c:128:check_vendorid(): received Vendor
 ID: KAME/racoon
2003-02-21 18:11:38: INFO: isakmp.c:2417:log_ph1established(): ISAKMP-SA es
tablished 192.168.1.100[500]-192.168.2.100[500] spi:6a01ea039be7bac2:bd288f
f60eed54d0
2003-02-21 18:11:39: INFO: isakmp.c:938:isakmp_ph2begin_i(): initiate new p
hase 2 negotiation: 192.168.1.100[0]<=>192.168.2.100[0]
2003-02-21 18:11:39: INFO: pfkey.c:1106:pk_recvupdate(): IPsec-SA establish
ed: ESP/Tunnel 192.168.2.100->192.168.1.100 spi=68291959(0x4120d77)
2003-02-21 18:11:39: INFO: pfkey.c:1318:pk_recvadd(): IPsec-SA established:
 ESP/Tunnel 192.168.1.100->192.168.2.100 spi=223693870(0xd554c2e)
```

X.509 Certificates

Racoon supports the usage of X.509 certificates for the authentication process. These certificates may be checked against a certificate authority (CA). The configuration is similar to the PSK configuration and differs only on the authentication part:

```
path certificate "/etc/certs";

remote 192.168.2.100 {
        exchange_mode main;
        certificate_type x509 "my_certificate.pem" "my_private_key.pem";
 verify_cert on;
        my_identifier asn1dn;
 peers_identifier asn1dn;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method rsasig;
                dh_group modp1024;
        }
}

sainfo address 172.16.1.0/24 any address 172.16.2.0/24 any {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
}
```

The certificate and the private key are stored in the certificate path `/etc/certs`. This path is set using the option **path certificate** in the configuration file. The certificates and the certificate revocation lists are stored in PEM format as generated with **openssl**. For the generation of certificates see the chapter on X.509 certificates. If the certificate of the peer is to be checked against a certificate authority (**verify_cert**

on; is the default), then the certificate of the CA has to be also stored in this directory. For OpenSSL to find the certificate it has to be renamed or linked using the hashed name:

```
ln -s CAfile.pem `openssl x509 -noout -hash < CAfile.pem`.0
```

If the certificate additionally is to be checked against a certificate revocation file (CRL) the CRL must be stored in the same directory using a similar linked hashed name:

```
ln -s CRLfile.pem `openssl x509 -noout -hash < CAfile.pem`.r0
```

When storing the certificates and the private key it is important to note that **racoon** cannot decrypt a private key. Therefore the private key must be stored in its decrypted cleartext form. If you created a crypted private key, you have to decrypt it:

```
# openssl rsa -in my_private_key.pem -out my_private_key.pem
read RSA key
Enter PEM pass phrase: password
writing RSA key
```

Roadwarrior

Roadwarriors are clients using unknown dynamic IP addresses to connect to a VPN gateway. In combination with **racoon** this poses two problems:

- The IP address is not known and cannot be specified in the **racoon** configuration file or in the `/etc/psk.txt` file. A different way to determine the identity of the client must be found. When using pre-shared keys this requires the aggressive mode! The best solution is the usage of X.509 certificates though.
- No security policy can be created for **racoon** to act on, since the destination IP address is not known. **racoon** must create the security policy and the security association when the connection is initiated.

To achieve this the configuration file `/etc/racoon.conf` needs several modifications:

```
path certificate "/etc/certs";

remote anonymous {
        exchange_mode main;
        generate_policy on;
        passive on;
        certificate_type x509 "my_certificate.pem" "my_private_key.pem";
        my_identifier asn1dn;
        peers_identifier asn1dn;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method rsasig;
                dh_group modp1024;
        }
}


sainfo anonymous {
        pfs_group modp1024;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
```

```
}
```

The option **generate_policy on** instructs **racoon** to create the appropriate policy when a new connection is initiated. The option **passive on** tells **racoon** to remain passive and wait for new connection to be started from the outside. **racoon** may not start a connection.

The most important change though is the definition of **anonymous** in the **remote** and **sainfo** line. This instructs **racoon** to accept the connection from anywhere.

NAT-Traversal

The Linux kernel 2.6 is capable of using NAT traversal in tunnel mode. Transport mode is not supported yet. This can be used by Racoon starting with version 0.3.3 of the ipsec-tools.

To configure Racoon for NAT traversal several options have been added to the configuration file. These are **natt_keepalive**, **isakmp_natt**, **nat_traversal**.

The most important option is **nat_traversal**. This can be set to **on**, **off** or **force**. When set to **on** this peer will use NAT traversal as soon as a NAT device is detected on the path. **Off** will disable this behavior. When using **force** NAT traversal will be used regardless wether a NAT device is found or not.

Since many NAT devices forget the entries in their internal tables quite fast when no traffic is seen, racoon offers to send keepalive packets across the wire. These are send every 20 seconds by default. You can change this value using **natt_keepalive**. Setting this to 0 seconds will disable this feature.

If you want to use NAT traversal you have to specify the IP address and the port to use in the **listen** section of the racoon configuration file. This is done using **isakmp_natt**.

For clarity a typical configuration file is shown, where the peer 192.168.2.100 is hidden by a NAT gateway with the IP address 192.168.1.1:

```
path pre_shared_key "/etc/psk.txt";

timer  {
       natt_keepalive 10sec;
       }

listen {
       isakmp 192.168.1.100 [500];
       isakmp_natt 192.168.1.100 [4500];
       }

remote 192.168.1.1 {
        exchange_mode main;
        nat_traversal on;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;
        }
}

sainfo address 172.16.1.0/24 any address 172.16.2.0/24 any {
        pfs_group modp768;
        encryption_algorithm 3des;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate;
```

```
}
```

If you configured everything correctly NAT will be detected automatically:

```
2004-12-22 07:34:53: INFO: @(#)ipsec-tools 0.4 (http://ipsec-tools.sourceforge.net)
2004-12-22 07:34:53: INFO: @(#)This product linked OpenSSL 0.9.7a Feb 19 2003 (http://www.openssl.org/)
2004-12-22 07:34:53: INFO: 192.168.1.100[4500] used as isakmp port (fd=6)
2004-12-22 07:34:53: INFO: 192.168.1.100[4500] used for NAT-T
2004-12-22 07:34:53: INFO: 192.168.1.100[500] used as isakmp port (fd=7)
2004-12-22 07:35:09: INFO: respond new phase 1 negotiation: 192.168.1.100[500]<=>192.168.1.1[500]
2004-12-22 07:35:09: INFO: begin Identity Protection mode.
2004-12-22 07:35:09: INFO: received Vendor ID: draft-ietf-ipsec-nat-t-ike-02
2004-12-22 07:35:09: INFO: received Vendor ID: RFC XXXX
2004-12-22 07:35:09: INFO: Selected NAT-T version: RFC XXXX
2004-12-22 07:35:09: INFO: Hashing 192.168.1.100[500] with algo #1
2004-12-22 07:35:09: INFO: NAT-D payload #0 verified
2004-12-22 07:35:09: INFO: Hashing 192.168.1.1[500] with algo #1
2004-12-22 07:35:09: INFO: NAT-D payload #1 doesn't match
2004-12-22 07:35:09: INFO: NAT detected: PEER
2004-12-22 07:35:10: INFO: Hashing 192.168.1.1[500] with algo #1
2004-12-22 07:35:10: INFO: Hashing 192.168.1.100[500] with algo #1
2004-12-22 07:35:10: INFO: Adding remote and local NAT-D payloads.
2004-12-22 07:35:10: INFO: NAT-T: ports changed to: 192.168.1.1[4500]<->192.168.1.100[4500]
2004-12-22 07:35:10: INFO: KA list add: 192.168.1.100[4500]->192.168.1.1[4500]
2004-12-22 07:35:10: INFO: ISAKMP-SA established 192.168.1.100[4500]-192.168.1.1[4500] spi:0613dc09c4ccc828:9cc9dfc9acc82eb5
2004-12-22 07:35:11: INFO: respond new phase 2 negotiation: 192.168.1.100[0]<=>192.168.1.1[0]
2004-12-22 07:35:11: INFO: Adjusting my encmode UDP-Tunnel->Tunnel
2004-12-22 07:35:11: INFO: Adjusting peer's encmode UDP-Tunnel(3)->Tunnel(1)
2004-12-22 07:35:11: INFO: IPsec-SA established: ESP/Tunnel 192.168.1.1->192.168.1.100 spi=95762109(0x5b536bd)
2004-12-22 07:35:11: INFO: IPsec-SA established: ESP/Tunnel 192.168.1.100->192.168.1.1 spi=222871470(0xd48bfae)
```

When looking at the packets on the wire you will see UDP traffic traversing back and forth:

```
[root@bibo root]# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap1, link-type EN10MB (Ethernet), capture size 96 bytes
13:37:41.920621 IP 192.168.1.1.isakmp > 192.168.1.100.isakmp: isakmp: phase 1 I iden
t
13:37:41.941296 IP 192.168.1.100.isakmp > 192.168.1.1.isakmp: isakmp: phase 1 R iden
t
13:37:42.051826 IP 192.168.1.1.isakmp > 192.168.1.100.isakmp: isakmp: phase 1 I iden
t
13:37:42.157134 IP 192.168.1.100.isakmp > 192.168.1.1.isakmp: isakmp: phase 1 R iden
t
13:37:42.353942 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 72
13:37:42.361530 IP 192.168.1.100.4500 > 192.168.1.1.4500: UDP, length 72
13:37:42.373799 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 88
13:37:43.374630 IP 192.168.1.100.4500 > 192.168.1.1.4500: UDP, length 1
13:37:43.384476 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 256
13:37:43.431219 IP 192.168.1.100.4500 > 192.168.1.1.4500: UDP, length 256
13:37:43.436680 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 56
13:37:44.492976 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 1
13:37:45.390137 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 116
13:37:45.390612 IP 192.168.1.100.4500 > 192.168.1.1.4500: UDP, length 116
13:37:46.395603 IP 192.168.1.1.4500 > 192.168.1.100.4500: UDP, length 116
13:37:46.396009 IP 192.168.1.100.4500 > 192.168.1.1.4500: UDP, length 116
```

If you are using your Racoon not in a roadwarrior setup but with fixed addresses as above you need to modify your Security Policies too. These need to reflect the natted addresses! The correct Policies for the above scenario are:

```
#!/usr/sbin/setkey -f
#
# Flush SAD and SPD
flush;
spdflush;

# Create policies for racoon
spdadd 172.16.1.0/24 172.16.2.0/24 any -P out ipsec
           esp/tunnel/192.168.1.100-192.168.1.1/require;

spdadd 172.16.2.0/24 172.16.1.0/24 any -P in ipsec
           esp/tunnel/192.168.1.1-192.168.1.100/require;
```

These policies are automatically setup if you use **generate_policy on;** in your Racoon configuration.

Linux Kernel 2.6 using OpenBSD's isakmpd

Thomas Walpuski has ported the IKE daemon of the OpenBSD operating system to Linux (http://bender.thinknerd.de/~thomas/IPsec/isakmpd-linux.html). The isakmpd can now be used on Linux kernel 2.5.47+ and 2.6.x to setup IPsec VPNs. This chapter will describe the installation and configuration of the isakmpd.

Installation

If you are using a RPM based distribution or Debian the installation may be done using the appropiate package tools. The author of this document has compiled an RPM package of the isakmpd for the Linux kernel 2.6.0 (http://www.spenneberg.org/VPN/Kernel-2_6_IPsec). Please be aware, that this package may not work on testversions, because the ABI in the kernel has been changed several times. The debian project includes a package which may be installed using **apt-get install isakmpd**.

When installing from source you need the keynote package (http://www1.cs.columbia.edu/~angelos/keynote.html) if you want to use X.509 certificates. Additionally you need a Linux kernel 2.5.47+ or 2.6.x.

To get the isakmpd sources follow the steps mentioned on the webpage of Thomas Walpuski. Then edit the GNUmakefile accordingly and activate the line **OS=linux**. If you are not keeping the Linux kernel in `/usr/src/linux` you will need to additionally modify the file `sysdep/linux/GNUmakefile.sysdep`.

The compilation may be done using the command **make**.

The isakmpd comes with two additional commands: **keyconv** and **certpatch**. These tools are in the subdirectory `apps` and may be compiled by hand (They are part of my RPM-package). **Certpatch** can add a SubjectAltName to an existing certificate while keyconv converts DNSSEC to openssl keys and vice-versa.

I was able to compile these tools successfully using (Your mileage may vary.):

```
gcc -DMP_FLAVOUR=MP_FLAVOUR_GMP -I../.. -I../../sysdep/linux   -I /usr/src/linux-2.6.0 -lcrypto -lgmp certpatch.c -o certpatch
gcc  -I../.. -I../../sysdep/linux   -I /usr/src/linux-2.6.0 -lcrypto -lgmp base64.c keyconv.c -o keyconv
```

One last caveat: All manpages are in Latin1 format. Red Hat 9 cannot display these manpages. You have to convert them to be able to read them (done in the RPM-package): **iconv --from-code LATIN1 --to-code UTF-8 --output isakmpd2.8 isakmpd.8**

When the isakmpd has been compiled, generate the mandatory directory structure:

```
mkdir /etc/isakmpd
mkdir /etc/isakmpd/ca
mkdir /etc/isakmpd/certs
mkdir /etc/isakmpd/keynote
mkdir /etc/isakmpd/crls
mkdir /etc/isakmpd/private
mkdir /etc/isakmpd/pubkeys
```

Using preshared keys (PSK)

The isakmpd uses one configuration file and one policy file. These are the file `/etc/isakmpd/isakmpd.conf` and `/etc/isakmpd/isakmpd.policy`. The configuration uses the well known format called .INI style. Each section starts with a line like:

```
[section]
```

Within the section you can assign a value to a tag:

```
tag=value
```

If the value is longer than one line you can use the Backslash technique to use several lines. Comments may be put anywhere using the hash mark #.

To start we will look at a simple configuration which uses a preshared secret for the authentication. Please take a look at Figure 5 for the setup.

```
[General]
Listen-on=              192.168.1.100

[Phase 1]
192.168.2.100=                ISAKMP-peer-west

[Phase 2]
Connections=            IPsec-east-west

[ISAKMP-peer-west]
Phase=                  1
Local-address=          192.168.1.100
Address=                192.168.2.100
Authentication=         ThisIsThePassphrase

[IPsec-east-west]
Phase=                  2
ISAKMP-peer=            ISAKMP-peer-west
Configuration=          Default-quick-mode
Local-ID=               Net-east
Remote-ID=              Net-west

[Net-west]
ID-type=                IPV4_ADDR_SUBNET
Network=                172.16.2.0
Netmask=                255.255.255.0

[Net-east]
ID-type=                IPV4_ADDR_SUBNET
Network=                172.16.1.0
Netmask=                255.255.255.0

[Default-quick-mode]
DOI=                    IPSEC
EXCHANGE_TYPE=          QUICK_MODE
```

```
Suites=                 QM-ESP-3DES-MD5-PFS-SUITE
```

This configuration file describes a tunnel between the two gateways 192.168.1.100 and 192.168.2.100. This tunnel may be used by 172.16.1.0/24 and 172.16.2.0/24. This configuration file is specifically for the gateway 192.168.1.100.

Let's look at the individual sections. The first section [General] describes the general setup. Here we define if isakmpd should bind to specific IP addresses during startup. This is recommended if you have several IP addresses on your VPN gateway.

The section [Phase 1] describes which configuration to use for the peer using the IP address 192.168.2.100. If the IP address of the peer is not known (roadwarrior) you can use default instead.

The section [Phase 2] describes the tunnels to create once a Phase 1 authentication has been established. If isakmpd may not actively start the connections use **Passive-connections** instead.

Now you have to define the names you referred to in the Phase 1 and Phase 2 sections. First we define the ISAKMP-peer-west. This definition is used in **Phase** 1 and we know the **Local-address** and the remote **Address**. If the remote address is not known, just remove this tag. **Authentication** should be done using a preshared key which is given in cleartext.

Next the tunnel IPsec-east-west is defined. It is used in **Phase** 2 and shall be established with the **ISAKMP-peer** ISAKMP-peer-west. We want to define the **Configuration** of the connection and the additional IDs for the tunnel (**Local-ID** and **Remote-ID**).

Since these IDs are referrals again, we have to define them. The **ID-type** may be IPV4_ADDR, IPV6_ADDR, IPV4_ADDR_SUBNET and IPV6_ADDR_SUBNET.

Last but not least we have to define the quick-mode configuration, we referred to in the description of the tunnel. We define the **DOI** (default: IPSEC), the **EXCHANGE_TYPE** (default: QUICK_MODE) and the **Suites** to use. This is QuickMode-Encapsulated-Security-Payload-3DES-Encryption-MD5-HMAC-Perfect-Forward-Secrecy. You can specify several suites seperated by commas. Read the man-page for all possible transforms and suites.

The `isakmpd.policy` file is much shorter. The next listing shows an example:

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "passphrase:ThisIsThePassphrase"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg == "3des" &&
            esp_auth_alg == "hmac-md5" -> "true";
```

For testing the connection start the isakmpd using the following line:

```
isakmpd -d -4 -DA=90
```

This will start the isakmpd in foreground (-d) using IPv4 (-4) and a debuglevel of 90.

Once the connection has started you should be able to ping from one subnet to the other subnet. If you have also installed the ipsec-tools you can use the command **setkey** to view the policies and security associations added by the isakmpd. If you kill the isakmpd running in foreground using ctrl-c, it does not flush the SAD and SPD. You will have to do this manually using the command **setkey**. If you kill the isakmpd using the command **kill -TERM** it will flush the SAD and SPD!

Using X.509 certificates

The isakmpd may also use X.509 certificates for the authentication process. You can create your certificates using the usual tools and need for each machine, taking part in the VPN, the following files:

- `/etc/isakmpd/private/local.key` The private key of the machine in .pem format. Permissions must be 600.
- `/etc/isakmpd/ca/ca.crt` The certificate of the certificate authority you trust.
- `/etc/isakmpd/certs/ip-address.crt` The certificate of the local machine.

For isakmpd to find and use the certificate it has to include a SubjectAltName. This X.509v3 extension can be defined during generation of the certificate or later using the command **certpatch**. This command needs the private key of the CA, extracts the certificate, adds the extension and signs the certificate again.

```
certpatch -i ip-address -k ca.key  originalcert.crt newcert.crt
```

Certpatch can add an IP address, a FQDN or a UFQDN to the certificate.

Once these files are stored in the appropiate folders and have the appropiate permissions assigned, you can create the configuration and the policy file. In the configuration file just remove the line Authentication. and add a line **ID=East** to the ISAKMP-peer-west section. Then define East. Additionally you have to specifiy the X.509 directories. The full configuration file follows:

```
[General]
Listen-on=              192.168.1.100

[Phase 1]
192.168.2.100=                ISAKMP-peer-west

[Phase 2]
Connections=            IPsec-east-west

[ISAKMP-peer-west]
Phase=                  1
Local-address=          192.168.1.100
Address=                192.168.2.100
ID=                     East

[East]
ID-type=                IPV4_ADDR
Address=                192.168.1.100

[IPsec-east-west]
Phase=                  2
ISAKMP-peer=            ISAKMP-peer-west
Configuration=          Default-quick-mode
Local-ID=               Net-east
Remote-ID=              Net-west

[Net-west]
ID-type=                IPV4_ADDR_SUBNET
Network=                172.16.1.0
Netmask=                255.255.255.0

[Net-east]
ID-type=                IPV4_ADDR_SUBNET
Network=                172.16..2.0
Netmask=                255.255.255.0

[Default-quick-mode]
```

```
DOI=                    IPSEC
EXCHANGE_TYPE=          QUICK_MODE
Suites=                 QM-ESP-3DES-MD5-PFS-SUITE
[X509-certificates]
CA-directory=           /etc/isakmpd/ca/
Cert-directory=         /etc/isakmpd/certs/
Private-key=            /etc/isakmpd/private/local.key
```

The policy file needs to be modified, too. Since you only want to allow peers using certificates signed by the trusted CA add the following line after the line Authorizer. The full policy file follows:

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "DN:/C=DE/ST=NRW/L=Steinfurt/O=Spenneberg.Com/OU=VPN/CN=RootCA"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg == "3des" &&
            esp_auth_alg == "hmac-md5" -> "true";
```

The text after **DN:** has to match the subject line of the CA certificate:

```
openssl x509 -in ca/ca.crt -noout -subject
```

Now you can start the isakmpd as usual to test the configuration.

Generating X.509 Certificates

Today almost all VPN implementations allow the usage of X.509 certificate for the authentication of the peers. These are the same certificates as used for the implementation of the Secure Socket Layer (SSL) in the HTTP protocol.

This chapter will briefly cover the creation of these certificates.

Using OpenSSL

The easiest way to create X.509 certificates on Linux is the **openssl** command and the auxiliary tools. When the OpenSSL package has been installed usually an auxillary command **CA** and/or **CA.pl**, has been installed, too. We will use this command to create the certificates.

First check where the command has been installed. It is usually not in your path! On Red Hat Linux distributions it is installed in `/usr/share/ssl/misc/CA`.

Now create your certificate authority first.

```
$ mkdir certs
$ cd certs
$ /usr/share/ssl/misc/CA -newca
CA certificate filename (or enter to create) <enter>

Making CA certificate ...
Using configuration from /usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
................++++++
..............++++++
writing new private key to './demoCA/private/./cakey.pem'
Enter PEM pass phrase: capassword
Verifying password - Enter PEM pass phrase: capassword
```

```
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name (full name) [NRW]:
Locality Name (eg, city) [Steinfurt]:
Organization Name (eg, company) [Spenneberg.com]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:RootCA 2003
Email Address []:ralf@spenneberg.net
```

Please enter the appropiate values when asked for Country Name, etc. If you would like to have the correct values proposed (like above in my case) edit your `openssl.cnf` file. On Red Hat Linux systems you may usually find it at `/usr/share/ssl/openssl.cnf`.

The created certificate authority is only valid for one year. Often you want a longer lifetime for the certificate of your CA. Since the certificates you are signing later on usually have a shorter lifetime it is not practical to edit the `openssl.cnf` file. Rather change the lifetime manually:

```
$ cd demoCA/
$ openssl x509 -in cacert.pem -days 3650 -out cacert.pem
-signkey ./private/cakey.pem
Getting Private key
Enter PEM pass phrase: capassword
$ cd ..
```

The certificate authority is now ready to go. Let's create a certificate signing request:

```
$ /usr/share/ssl/misc/CA -newreq
Using configuration from /usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
..............................++++++
.................................++++++
writing new private key to 'newreq.pem'
Enter PEM pass phrase: certpassword
Verifying password - Enter PEM pass phrase: certpassword
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [DE]:
State or Province Name (full name) [NRW]:
Locality Name (eg, city) [Steinfurt]:
Organization Name (eg, company) [Spenneberg.com]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:VPN-Gateway
Email Address []:ralf@spenneberg.net

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Request (and private key) is in newreq.pem
```

The file `newreq.pem` contains the certificate signing request and the encrypted private key. This file can later be used as a private key for FreeS/WAN or Racoon. Once the request is created, we can sign it using the certificate authority.

```
$ /usr/share/ssl/misc/CA -sign
Using configuration from /usr/share/ssl/openssl.cnf
Enter PEM pass phrase: capassword
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'DE'
stateOrProvinceName   :PRINTABLE:'NRW'
localityName          :PRINTABLE:'Steinfurt'
organizationName      :PRINTABLE:'Spenneberg.com'
commonName            :PRINTABLE:'VPN-Gateway'
emailAddress          :IA5STRING:'ralf@spenneberg.net'
Certificate is to be certified until Apr 29 06:08:56 2004 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Depending on the version of the command **CA** the certificate might be print to stdout. This will be similar to the following certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=DE, ST=NRW, L=Steinfurt, O=Spenneberg.com,
CN=RootCA 2003/Email=ralf@spenneberg.net
        Validity
            Not Before: Apr 30 06:08:56 2003 GMT
            Not After : Apr 29 06:08:56 2004 GMT
        Subject: C=DE, ST=NRW, L=Steinfurt, O=Spenneberg.com,
CN=VPN-Gateway/Email=ralf@spenneberg.net
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c5:3b:9c:36:3a:19:6c:a9:f2:ba:e9:d2:ed:84:
                    33:36:48:07:b2:a3:2d:59:92:b0:86:4c:81:2c:ea:
                    5c:ed:f3:ba:eb:17:4e:b3:3a:cc:b7:5b:5d:ca:b3:
                    04:ed:fb:59:3c:c5:25:3e:f3:ff:b0:22:10:fb:de:
                    72:0a:ee:42:4b:9a:d3:27:d3:b6:fb:e9:88:10:c8:
                    47:b7:26:4f:71:40:e4:75:c4:c0:ee:6b:87:b8:6f:
                    c9:5e:66:cf:bb:e7:ad:72:68:b8:6d:fd:8f:4c:1f:
                    3a:a2:0d:43:25:06:b9:92:e7:20:6c:86:15:a0:eb:
                    7f:f7:0b:9a:99:5d:14:88:9b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                CB:5C:19:9B:E6:8A:8A:FE:0E:C4:FD:5E:DF:F7:BF:3D:A8:
18:7C:08
            X509v3 Authority Key Identifier:
                keyid:01:BB:C6:33:BE:F5:9A:5E:B0:0C:5D:BD:41:E9:78:
```

```
6C:54:AD:66:8E
                DirName:/C=DE/ST=NRW/L=Steinfurt/O=Spenneberg.com/
CN=RootCA 2003/Email=ralf@spenneberg.net
                serial:00

    Signature Algorithm: md5WithRSAEncryption
        6f:89:2b:95:af:f1:8d:4d:b7:df:e8:6d:f7:92:fb:48:8c:c4:
        1a:43:68:65:97:01:87:a6:84:b5:a1:38:bd:62:74:70:db:9e:
        78:19:d9:0c:af:18:ad:13:77:56:7d:3f:19:61:da:ba:74:30:
        8e:c5:50:0e:e3:eb:ff:95:cd:8d:d6:7e:c3:0e:ab:5b:34:94:
        bc:16:0f:ef:dc:de:40:bb:7d:ba:a2:b8:5d:f9:74:e7:28:58:
        75:a0:66:d2:8d:85:ba:38:82:08:10:33:ef:be:29:c9:31:9d:
        63:a9:f7:e0:99:ea:a7:ed:b6:b5:33:1b:1c:4a:a4:05:40:6e:
        40:7b
-----BEGIN CERTIFICATE-----
MIIDjDCCAvWgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBgjELMAkGA1UEBhMCREUx
DDAKBgNVBAgTA05SVzESMBAGA1UEBxMJU3RlaW5mdXJ0MRcwFQYDVQQKEw5TcGVu
bmViZXJnLmNvbTEUMBIGA1UEAxMLUm9vdENBIDIwMDMxIjAgBgkqhkiG9w0BCQEW
E3JhbGZAc3Blbm5lYmVyZy5uZXQwHhcNMDMwNDMwMDYwODU2WhcNMDQwNDI5MDYw
ODU2WjCBgjELMAkGA1UEBhMCREUxDDAKBgNVBAgTA05SVzESMBAGA1UEBxMJU3Rl
aW5mdXJ0MRcwFQYDVQQKEw5TcGVubmViZXJnLmNvbTEUMBIGA1UEAxMLVlBOLUdh
dGV3YXkxIjAgBgkqhkiG9w0BCQEWE3JhbGZAc3Blbm5lYmVyZy5uZXQwgZ8wDQYJ
KoZIhvcNAQEBBQADgY0AMIGJAoGBAMU7nDY6GWyp8rrp0u2EMzZIB7KjLVmSsIZM
gSzqXO3zuusXTrM6zLdbXcqzBO37WTzFJT7z/7AiEPvecgruQkua0yfTtvvpiBDI
R7cmT3FA5HXEwO5rh7hvyV5mz7vnrXJouG39j0wfOqINQyUGuZLnIGyGFaDrf/cL
mpldFIibAgMBAAGjggEOMIIBCjAJBgNVHRMEAjAAMCwGCWCGSAGG+EIBDQQfFh1P
cGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0ZTAdBgNVHQ4EFgQUy1wZm+aKiv4O
xP1e3/e/PagYfAgwga8GA1UdIwSBpzCBpIAUAbvGM771ml6wDF29Qel4bFStZo6h
gYikgYUwgYIxCzAJBgNVBAYTAkRFMQwwCgYDVQQIEwNOUlcxEjAQBgNVBAcTCVN0
ZWluZnVydDEXMBUGA1UEChMOU3Blbm5lYmVyZy5jb20xFDASBgNVBAMTC1Jvb3RD
QSAyMDAzMSIwIAYJKoZIhvcNAQkBFhNyYWxmQHNwZW5uZWJlcmcubmV0ggEAMA0G
CSqGSIb3DQEBBAUAA4GBAG+JK5Wv8Y1Nt9/obfeS+0iMxBpDaGWXAYemhLWhOL1i
dHDbnngZ2QyvGK0Td1Z9Pxlh2rp0MI7FUA7j6/+VzY3WfsMOq1s0lLwWD+/c3kC7
fbqiuF35dOcoWHWgZtKNhbo4gggQM+++KckxnWOp9+CZ6qfttrUzGxxKpAVAbkB7
-----END CERTIFICATE-----
Signed certificate is in newcert.pem
```

It is now advisable to rename the files `newreq.pem` and `newcert.pem` to something more meaningful.

```
$ mv newcert.pem vpngateway_cert.pem
$ mv newreq.pem vpngateway_key.pem
```

Now have fun creating certificates for every peer in the VPN.

In case a private key gets stolen or compromised, you have to revoke it because based on its lifetime it is still valid. The revoked keys are stored in the certificate revocation list (CRL). First, create an (empty) list:

```
$ openssl ca -gencrl -out crl.pem
Using configuration from /usr/share/ssl/openssl.cnf
Enter PEM pass phrase: capassword
```

You need to create the file `democA/crlnumber` manually if you get an error: No such file. Modern OpenSSL versions require this.

```
$ echo 01 > demoCA/crlnumber
```

To revoke a certificate you need to have the certificate file. This is also stored in `demoCA/newcerts/`. The name of the certificate can be read in `demoCA/index.txt`. Then use the following command.

```
$ openssl ca -revoke compromised_cert.pem
Using configuration from /usr/share/ssl/openssl.cnf
Enter PEM pass phrase: capassword
Revoking Certificate 01.
Data Base Updated
```

Once the certificate has been revoked, the certificate revocation list has to be recreated using the above command.

Generating Certificates for Windows Clients

When generating certificates for Windows clients you have to make sure that the lifetime of the certificate lies within the lifetime of the CA. If the lifetime of the certificate exceeds the lifetime of the CA, the windows client will not accept the certificate!

The easiest way to transfer certificates to a windows box is by using the PKCS#12 exchange format. Openssl can reformat the certificates to this format:

```
$ openssl pkcs12 -export -inkey key.pem -in cert.pem -certfile cacert.pem -out export.p12 -name "Windows Cert"
```

You are asked to specify an export password. On the windows box you can then import this file using the export password.

A tool which might help in generating the PKCS#12-File is Wincert. You find the URL to the tool in the links section.

Advanced Configuration

Xauth and IKE-Mode-Config

Unfortunately Xauth and IKE-Mode-Config are broken on Linux using ipsec-tools <= 0.6. As soon as Xauth is working I will put up some hints on using these.

IPtables Rules

Using ESP in tunnel mode without compression increases the size of the packets transferred. This even happens sometimes when compression is activated. This can cause problems in tunnel mode when the client does not know that the packet is to be encapsulated. If the clients sends a packet of 1500 bytes the additional encapsulation will increase the size of the packet. For TCP-packets you can solve this problem setting the MSS on both sides of the tunnel using the **iptables** command:

```
iptables -t mangle -A PREROUTING -p esp -j MARK --set-mark 1
iptables -A FORWARD -m mark --mark 1 -p tcp --tcp-flags SYN,RST SYN -j
TCPMSS --set-mss 1400
```

Links

This section will provide some links to tools you might need.

- IPsec-Tools: http://ipsec-tools.sf.net
- Markus Mueller ipsec.exe to connect Windows machines to the VPN: http://vpn.ebootis.de
- Wincert helps in the generation of PKCS#12-Certificate files: http://sourceforge.net/projects/wincert/

Notes

1. http://www.tldp.org
2. http://www.ipsec-howto.org
3. http://www.tldp.org/HOWTO/Networking-Overview-HOWTO.html
4. http://www.tldp.org/HOWTO/Net-HOWTO/index.html
5. http://www.tldp.org/HOWTO/VPN-Masquerade-HOWTO.html
6. http://www.tldp.org/HOWTO/VPN-HOWTO/
7. http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/
8. http://www.kernel.org
9. http://ipsec-tools.sourceforge.net
10. http://bender.thinknerd.de/~thomas/IPsec/isakmpd-linux.html
11. http://www.spenneberg.org/VPN/Kernel-2_6_IPsec
12. http://www1.cs.columbia.edu/~angelos/keynote.html
13. http://ipsec-tools.sf.net
14. http://vpn.ebootis.de
15. http://sourceforge.net/projects/wincert/