Chapter 1

Introduction

1.1 Why?

The old VPN solution was unreliable, relied on proprietary software and only supported Windows computers.

1.2 Goals

One of the primary goals of this VPN-Solution is compatibility. There must be VPN-clients for every operating system that the company uses, so the main focus in this case is Windows, Mac OS X, and Linux. Additionally, handhelds should be able to connect to the VPN. The preferred solution would not require a proprietary VPN-Client, but rather utilize the built-in clients of the operating systems. As the goal of this project thesis is to use only free and open source software, the VPN services should only rely on free implementations of VPN-protocols, and operating systems.

The operating system of my choice was Linux 2.6, for several reasons:

- I know it
- I like the way, most Linux distributions deal with packages
- Linux is widely supported
- Linux is free
- You can choose between ISAKMP Daemons[1], and are not forced to use the BSD-Implementation, if you took a BSD, for example.
- Linux usually tries to support interoperability, whereas BSD usually doesn't care about interoperability too much.
- You can find a lot of documentation about Linux on the Internet.

1.3 Expectations for the reader

This document describes a technical implementation for an IPsec based VPN solution. It deals with the server and the client implementation, and tries to be a guide to make the reader understand why and how I did things. It is deeply technical, and a good or better deep knowledge about Linux and networking is assumed.

[1]More on ISAMKP later

Chapter 2

Technical Implementation

2.1 Standard Protocols

In this document we will deal with only 2 protocols, PPTP and IPSEC. Both are supported 'out of the box' by the Windows and the Macintosh (version 10) operating systems, and that is, why they are called 'standard' here.

2.1.1 PPTP

The Point to Point Tunneling Protocol (PPTP) is used to secure PPP connections over a TCP/IP link. Microsoft released extensions to the PPTP authentication mechanism (MS-CHAPv2). Another Microsoft extension is the *Microsoft Point to Point Encryption (MPPE).* It should be noted, that these extensions are Microsoft-proprietary, nevertheless, we will have a short look at them.

MS-CHAPv2 uses some very complicated hashing-algorithms, and some senseless procedures like using the SHA-Algorithm on some data twice.

MPPE is an encryption algorithm, that is only as good as the passwords that are used for authentication as MPPE keys get derived from MS-CHAPv2.

As all this is **very** Microsoft-proprietary, it is hard to implement solutions in open source software (although there are Linux-Kernel Patches), beyond that fact that the encryption algorithm is not secure, as it is only based on the authentication data. PPTP is therefore not the protocol we are looking for.

2.1.2 IPsec

IPsec, which is the short form for IP Security, is a set of protocols developed by the IETF to support secure exchange of packets at the IP layer. It's main purpose is to support VPN, which is exactly, what we are looking for. For IPsec to work, the sending and receiving devices must share a public key. This is a accomplished through a protocol known as *Internet Security Association and Key Management Protocol (ISAKMP)*, which allows the receiver to obtain a public key and authenticate the sender using digital certificates.

There are two modes of IPsec - tunnel and transport mode. Tunnel mode is used for secure communication between subnets, this is very useful for connecting two company-sites, for example. The solution we are looking for is called Transport mode. IPsec should guarantee, that the packets are encrypted, authenticated and anti-replay protected. IPsec encrypts every IP packet which requires that it reside in the kernel of an operating system, and not in userspace.

By design, IPsec communication is encrypted by symmetric algorithms (Blowfish, DES, 3DES). The whole packet is not encrypted, only the data (payload), not the IP-Headers. This is known as *ESP-Mode.* encrypted (and authenticated) headers can be achieved with *Authentication Header (AH)* mode. We will only focus on ESP-Mode, as we are interested in encrypting our data traffic, and are not interested in authentication in this layer.

Authentication can be done with a pre-shared key (which is considered weak) or by using certificates, which is quite a secure solution.

IPsec is our first choice, as it is an approved *IETF* standard, free implementations exist, and Windows, Mac, Pocket PC, Linux and BSD clients support it.

2.1.2.1 Kernel Space and Userspace

The IPsec Stack itself is implemented in the kernel space.

ISAKMP usually happens in userspace, this is where racoon or openswan come into play. Both, racoon and openswan are ISAKMP daemons. They deal with either pre-shared keys, or certificates, listen on UDP Port 500 and 4500 (optional for NAT-T), interact with the filesystem, and are necessary for the handshake. As soon as the encryption mechanisms and handshakes have been made, the kernel will communicate with the the corresponding side.

2.1.2.2 A word on NAT-Traversal

If a client is behind a NAT-firewall, IP packets are modified by the NAT-Device. This means that checksums concerning the IPsec packets would be incorrect. IPsec was originally not designed to work with NAT. Therefore NAT-Traversal extensions have been developed, to make IPsec work behind NAT devices. Clients and VPN-Server have to support NAT-T. Linux 2.6 supports NAT-T out of the box, whereas racoon at the time of writing supports only NAT-T in tunnel mode. Openswan supports tunnel and transport mode NAT-T, which is why openswan is the solution we are looking for.

2.2 Operating System clients

Choosing IPsec as the standard is a good start, but when it gets to the operating system built-in clients, things get more complicated.

2.2.1 Microsoft Windows XP/2000

Microsoft did some weird things with their VPN-standard-configuration. Microsoft claims, this is a standard, but in fact, their implementation is only based on standards. In Microsoft's implementation PPP packets are encapsulated in L2TP packets, which are encapsulated in IPsec packets. It is acceptable to use PPP for establishing a point-to-point connection, and using IPsec to establish a secure connection. But the value of L2TP in Microsoft's implementation is questionable. L2TP would come in very handy for Layer 3 independent protocols (not only IP could be transported over L2TP, as L2TP means *Layer 2 Tunneling Protocol*), however this doesn't make any sense, because PPP does not support this. Nevertheless, to stay compatible, we will use the same standards on the server side[1]. If this paragraph confused you, have a look at the illustrations in 2.6.

Beyond that protocol-encapsulation-jungle, Microsoft authenticates users by using certificates for IPsec, and additionally, using the *CHAP* Protocol. IPsec authentication using a

[1]Hint: It is possible to use pure IPsec with windows, without L2TP and PPP, but as you couldn't use the standard VPN-client but a console-tool, we will avoid this solution due to end-user confusion. Security doesn't suffer by using the PPP-L2TP-IPsec Solution, as IPsec is the outermost protocol, that will be actually transported over the wire.

pre-shared key is supported, but is considered insecure, and therefore is unacceptable . It is possible to use the pure IPsec stack of Windows, but you can't use the standard client then, which would confuse users.

2.2.2 Mac OS X 10.3.x

Mac OS X uses the same technology as Microsoft does. They do this to stay compatible with Microsoft, and therefore support connections to Windows-VPN-Servers. So all of 2.2.1 also applies to the Macintosh Operating System. Yet Mac OS X offers one great advantage: If you know racoon (will be described in the server section), you can tweak the config-file of the mac to your needs, which will get very handy later on.

2.2.3 Handhelds

Windows Handhelds with the Pocket PC 2003 operating system are fully supported. The configuration on the handheld computer is somewhat strange, because you can not just 'dial' in the VPN, because the handheld tries to be that smart, and decide for you, when to dial into the VPN. You have to create a special list for the handheld to let it know: "This is inside the VPN-network".

2.2.4 Unix Clients

Unix clients will work. Just read on.

2.3 Server Side

Portability is everything. I chose Linux for my server because I like the way Linux handles packages, and most if not all of the packages are available for Linux. Packages alleviate the need for manual compilation and make upgrades easy.

The IPsec Stack used, is based on KAME, which is available for most free UNIX-Operating Systems. You will find that the code is available on 2.6sec (Linux 2.6), FreeBSD and OpenBSD.

2.3.1 Not covered here

- Implementation with FreeSwan or StrongSwan. This configuration is possible with these products, yet it is not very portable.
- OpenVPN - a free userland-VPN implementation, which is quite good and very portable, but does not meet my requirements because it is not an IETF standard, does not use the built-in clients and does not support pocket devices.

2.3.2 Prerequisites

I will not go into the details of kernel compilation. There are good howtos on the Internet for that. You want to have IPsec configured in your kernel. In Linux 2.6 this would look similar to this:

```
CONFIG_INET_ESP=y
CONFIG_NET_KEY=y
CONFIG_CRYPTO_HMAC=y
CONFIG_CRYPTO_MD5=y
```

```
CONFIG_CRYPTO_SHA1=y
CONFIG_CRYPTO_DES=y
CONFIG_UNIX98_PTYS=y
CONFIG_LEGACY_PTYS=y
```

I do not recommend enabling modules in the kernel, unless you have a good reason (although I couldn't think of any). Note: You don't need any extra patches for your Linux kernel in this configuration (as opposed to ms-pptp)

Beyond the kernel configuration, be sure that your OS is up and running, and network-connections are working fine.

2.3.3 Software packages

- PPP is handled by pppd
- L2TP is handled by l2tpd (www.l2tpd.org)
- IPsec is handled by Linux 2.6 and openswan as the ISAKMP daemon (www.openswan.org)
- Alternatively, you can use racoon as the ISAKMP daemon (www.kame.net/racoon/)
- Certificates can be issued with openssl

Of course, all of the dependencies of these packages have to be met. ISAKMP is necessary to communicate with the kernel, whereas various implementations of l2tp daemons exist. Certificates do not necessarily need to be issued by openssl, and there might be an alternative to pppd. I chose the most supported software packages in this case.

2.3.4 Configuration tasks

2.3.4.1 openssl [Lnk: NCa]

To generate certificates, it is necessary to have a *certificate authority (CA)*. Setting up a certificate authority works pretty simple, if you use a shellscript, that is provided with openssl. In my case this shellscript resides in */usr/share/ssl/misc/* and is called CA. The command

```
./CA -newca
```

will create a new CA.

Every client that wants to connect to the VPN-box, will require a certificate. The VPN-box itself needs a certificate. You can do this with this two really simple commands:

```
./CA -newcert
./CA -signcert
```

In order to keep things organized, rename or copy the key-file and the cert-file:

```
cp newcert.pem workstation1.pem
cp newreq.pem workstation1.key
```

In order to use the certificate with racoon, it is necessary to strip the password from the private key-file, as racoon does not support private key-files, which will require a password. Do that with

```
openssl rsa -in workstation1.key -out workstation1.decrypted.key
```

As windows-clients do not like the pem-format, we export the certificates for the windows-clients to the p12 format (the p12 contains both, private key and certificate-file), as this format can be imported easily in windows:

```
openssl pkcs12 -export -in workstation1.pem -inkey \
workstation1.key -cert-file demoCA/cacert.pem -out \
workstation1.p12
```

Note that in the above example, it is assumed, that the root certificate resides in ./demoCA/cacert.pem.

2.3.4.2 openswan [Lnk: OSW] [Lnk: NCa]

Racoon has some problems, but there are some reasons why you might want to stick with it:

- You don't want to use Linux as your VPN-Server but a BSD-variant, for example.
- You don't care about NAT-Traversal issues in Transport Mode with racoon.
- NAT-T support for Racoon is already implemented, when you read this.

Otherwise Openswan is the ISAKMP daemon I chose. Several reasons led to this decision:

- Support for Certificate Revocation Lists (CRL) to revoke certificates
- Easy configuration Syntax
- Support for NAT-Traversal (Road Warriors behind NATed networks)
- Good documentation and support can be found on the Internet
- My personal experience with openswan: more rock-solid than racoon

Openswan does nothing differently compared to racoon, it is an *ISAKMP daemon*. The wonderful thing about openswan and Linux 2.6 is, that you don't need to patch your kernel to work with openswan. Openswan is fully compatible to the 2.6sec implementation. All you need to get openswan working, is the user-land daemon itself. If you want to use kernel 2.4 and openswan, you have to patch the kernel with the openswan kernel patches.

The creation of the config-file requires some tweaking and patience, and some help from the Internet [Lnk: JDL].

My sample configuration file for support for various road warriors looks like this (/etc/ipsec.conf):

```
version 2.0config setup
        interfaces=%defaultroute
        forwardcontrol=yes
        nat_traversal=yes
        virtual_private=%v4:10.0.0.0/8,\
              %v4:172.16.0.0/12,%v4:192.168.0.0/16
conn %default
        keyingtries=1
        compress=yes
        disablearrivalcheck=no
```

```
              authby=rsasig
              leftrsasigkey=%cert
              rightrsasigkey=%cert
              pfs=no
     conn roadwarrior-l2tp-updatedwin
              pfs=no
              leftprotoport=17/1701
              rightprotoport=17/1701
              also=roadwarrior
     conn roadwarrior-l2tp
             pfs=no
             leftprotoport=17/0
             rightprotoport=17/1701
             also=roadwarriorconn
     macintosh-l2tp
              pfs=no
              leftprotoport=17/1701
              rightprotoport=17/%any
              also=roadwarriorconn
     roadwarrior
              left=%defaultroute
              leftcert=vpn1.company.net.pem
              right=%any
              auto=add
              type=transport
```

Very Important: Indentations matter. Therefore do a tab-indentation for the indented lines, if you would copy my sample config-file.

- Left and right defines server-side and client-side. Left in my example stands for the server, and right for the clients.
- The *auto=add* configuration parameter enables the configuration section. Without this, you would be forced to 'start' this section by hand, which would not really become handy on a production server.
- The *also* configuration-parameter includes additional sections (works like #include in c).
- Openswan is able to identify the different clients on the behavior, *how* they send packets. It is visible in my example, that macintosh computer behave different than windows-computers.
- *PFS* stands for perfect forwarding secrecy, which would enable encryption algorithms, where penetration of the key-exchange protocol would not compromise keys negotiated earlier. Unfortunately, Windows does not support this without hacking the registry. [2]

Certificate files have to be copied to /etc/ipsec.d/ into the appropriate directories. Openswan also supports a password-protected private key file, but you have to type that secret password in a file (/etc/ipsec.secret) in clear-text anyway.

Openswan requires the ipsec-tools. It is poorly documented, but one has to flush the SA entries in the kernel after starting openswan. To start openswan, one might enter the following command in order to maintain connectivity to the openswan box:

[2]How often do we read this?

```
/etc/init.d/ipsec start; sleep 5; setkey -FP
```

Of course this has to be tweaked in the startup-scripts, too. Important: Openswan logs to /var/log/secure, not /var/log/daemon or similar files.

2.3.4.3 racoon [Lnk: KME]

Racoon is the userspace daemon for key exchange, speaks ISAKMP, and communicates from userland with the kernel to configure some parameters of IPsec. Racoon has to be run as a daemon to handle the correct key exchange. It will log to syslog for debugging, and is controlled by a few config files. Racoon will listen on UDP port 500, so the firewall in front of the VPN-box needs to allow traffic to this port.

The need for the user space daemon is quite simple: Racoon searches for certificates, that are provided via config file, it defines and restricts the key exchange modes between IPsec hosts, and sets key policies. Racoon is controlled over a config-file. My sample config-file is provided here:

```
log debug;
path certificate "/etc/racoon/certs/";
listen {
        isakmp 192.168.0.1 [500];
}
padding
{
        maximum_length 20;      # maximum padding length.
        randomize off;          # enable randomize length.
        strict_check off;       # enable strict check.
        exclusive_tail off;     # extract last one octet.
}
remote anonymous {
        exchange_mode main,aggressive;
        doi ipsec_doi;
        situation identity_only;
        generate_policy on;
        my_identifier asn1dn;
        peers_identifier asn1dn;
        verify_identifier on;
        certificate_type x509 "vpnbox.certificate.pem"  "decrypted.rsa.key";
        verify_cert off;
        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group modp1024;
        }
}
sainfo anonymous {
        lifetime time 28800 sec;
        encryption_algorithm 3des ;
        authentication_algorithm hmac_md5;
        compression_algorithm deflate ;
}
```

Every detail about this configuration can be found in the racoon man page, yet here some important hints:

- The default racoon.conf deals with pre-shared keys. Don't forget to insert the certificate path in the first line.
- The *remote* directive specifies the parameters for IKE phase 1 negotiation.
- Verifying the certificate only works, if the root certificate is signed by a trusted authority (e.g. Verisign, Thawte, etc). Therefore I had to turn *verify_cert* to *off*.
- Encryption algorithms are limited to the supported client encryption algorithms.
- The certificate file and the (passwordless) key file generation were described on Page 6 (openssl).
- The *sainfo* directive specifies the parameters for IKE phase 2 negotiation.

A script, that sets the IPsec policy, that will encrypt outgoing packets (for l2tpd) looks like this (I called it /etc/racoon/setup.sh):

```
#!/bin/bash
/sbin/setkey -FP
/sbin/setkey -F
/sbin/setkey -c << EOF
spdadd 192.168.0.1[1701] 0.0.0.0/0[0] any
   -P out ipsec esp/transport//require;
EOF
```

Hint: man setkey reveals all possible configuration data.

Start this script at boot-time, to set the IPsec Policy to encrypt for all outgoing data from the l2tpd. This script will actually also be called, when pppd hangs up, to reset the policy database (flush it).

2.3.4.4 l2tpd

L2tpd's task is to decapsulate the layer 2 packets (which are actually only ppp packets), and send them to pppd. The used l2tpd is quite a simple program, also the configuration is pretty simple. As mentioned before, l2tpd has no real use in this configuration, but it is necessary to use it, as otherwise the client connection will fail. l2tpd has no encryption or security functions in this case. It's main use would be to stay layer 3 protocol independent, which in this case, does not make sense. Nevertheless it provides the IP assignment for the client (as seen below).

The configuration file of l2tpd looks like this:

```
[global]
port = 1701
[lns default]
ip range = 192.168.0.10 - 192.168.0.20
local ip = 192.168.0.1
require chap = yes
refuse pap = yes
require authentication = yes
hostname = LinuxVPNserver
```

```
ppp debug = yes
pppoptfile = /etc/ppp/options.l2tpd
length bit = yes
```

Unfortunately there is almost no documentation concerning l2tpd.conf. I found this statement from the author (found in l2tpd.conf.sample):

```
; This example file should give you some idea of how the options for l2tpd
; should work.  The best place to look for a list of all options is in
; the source code itself, until I have the time to write better documentation :)
; Specifically, the file "file.c" contains a list of commands at the end.
```

2.3.4.5 pppd

The *Point to Point Protocol Daemon* decapsulates the ppp packets, and places them in the kernels IP stack. PPPD is also able to do authentication. This is what the Clients do. First they initiate a IPsec-connection. After a successful IPsec authentication, the CHAP process takes place, which authenticates against a password database. Luckily, PPPD is mature, and can authenticate against various databases - a plain text file with authentication data in it, the Linux passwd / shadow files, even ldap or radius would be supported.

The sample configuration for *options.l2tpd*[3] for me looks like this:

```
ipcp-accept-local
ipcp-accept-remote
ms-dns  192.168.0.9
ms-wins 192.168.0.9
auth
crtscts
idle 1800
mtu 1400
mru 1400
nodefaultroute
nodetach
debug
lock
proxyarp
connect-delay 5000
disconnect /etc/racoon/setup.sh
```

The man page of pppd describes all the details about these configuration directives. The last line with the disconnect statements flushes the SAD entries in the IPsec stack, and sets the policy up again.

At the moment I authenticate against a plain password file called chap-secrets:

```
# Secrets for authentication using CHAP
# client        server  secret                  IP addresses
wogri           *       "mypassword"            192.168.0.0/24
```

[3]This file is mentioned in the l2tpd.conf file, and defines the options for pppd

2.3.4.6 kernel configuration changes

- Enable IP forwarding via sysctl - here the Linux-example - sysctl.conf:

  ```
  # Controls IP packet forwarding
  net.ipv4.ip_forward = 1
  ```

- Disable source route verification

  ```
  # Controls source route verification
  net.ipv4.conf.default.rp_filter = 0
  ```

The enabling of IP forwarding is self explainitory. Source route verification must be turned off in order for packets that reach the interface to not dropped by the kernel. Route verification is the practice of discarding packets received on an interface which does not expect to those handle traffic from the given source address.

2.4 Client Side [Lnk: JDL]

The whole setup describes a *road warrior*[4] setup. The goal is to tell the client to use PPP/L2TP/IPsec encapsulation to send it's packets.

2.4.1 Microsoft Windows XP SP2

The following instructions apply to Windows XP SP2, whereas I assume all these rules also apply to XP SP1, XP and Windows 2000.

1. Go to network connections
2. Create a new network connection
3. Choose 'connect to the network at my workplace'
4. Choose VPN connection
5. Enter Company name
6. Choose to dial an initiating connection, if you have dial-up
7. Enter IP or hostname for your VPN Box
8. Finish

One might want to go to connection properties, networking tab and select L2TP IPsec VPN in the drop-down box (This speeds up your first connection try). Next step is to import the generated .p12 certificates on every windows-machine.

1. Open an mmc: Start | Run | type mmc | hit ENTER
2. Click File | Add/Remove Snap-in

[4]A road warrior is usually a laptop, that is 'on the road', and connects to the VPN from anywhere. These mobile computers have to be secured very well, as they might pose an invisible threat to the network if someone gains access to these laptops

3. Click Add
4. Select Certificates and click Add
5. Select Computer Account[5], click Next
6. Click Finish
7. Click Close
8. Click OK
9. Double Click Certificates
10. Right Click Personal, select All Tasks and click Import
11. Click Next
12. Enter the path to the .p12 file
13. Click Next
14. We stripped the password, so just click Next
15. Select 'Automatically select the certificate store based on the type of certificate', click Next
16. Click Finish

Done. No need to associate the Certificate with the Connection, as this is done automatically by Windows. In my project I automated this task, so that the clients would only get a setup.exe, the certificates would be downloaded from a secure server during the setup, and all other clickety stuff is done auto-magically (I used the Nullsoft installer and the Windows Scripting-Program Automateit for that).

2.4.2 Mac OS X

The Mac is based on config and log files, racoon, and a GUI wrapper, which makes configuration very easy.

Setup a VPN(L2TP) connection:

- Open Internet Connect
- Click File | New
- Choose L2TP over IPsec
- Enter the server address, account name (user name) and password

The next step is to tweak /etc/racoon/racoon.conf because the default on the Mac is that when you connect to a L2TP/IPsec host, the config file will be generated on the fly, and racoon will be started. As special parameters are needed, my racoon.conf looks like the following for a racoon-VPN-server (This configuration ignores the on the fly-config-files):

[5]It is important to import the certificate for the computer, as windows doesn't use the certificate otherwise. If you do only want the certificate be based on user-level, you will have to figure out some other method

```
path include "/etc/racoon" ;
# racoon will look for certificate file in the directory,
# if the certificate/certificate request payload is received.
path certificate "/etc/racoon/certs" ;
# "padding" defines some parameter of padding.  You should not touch
these.
padding
{
         maximum_length 20;      # maximum padding length.
         randomize off;          # enable randomize length.
         strict_check off;       # enable strict check.
         exclusive_tail off;     # extract last one octet.
}
listen
{

}
timer
{
         # These value can be changed per remote node.
         counter 10;             # maximum trying count to send.
         interval 3 sec; # interval to resend (retransmit)
         persend 1;              # the number of packets per a send.
         # timer for waiting to complete each phase.
         phase1 30 sec;
         phase2 30 sec;
}
sainfo address ::1 icmp6 address ::1 icmp6
{
         pfs_group 1;
         lifetime time 60 sec;
         encryption_algorithm 3des, cast128, blowfish 448, des ;
         authentication_algorithm hmac_sha1, hmac_md5 ;
         compression_algorithm deflate ;
}
remote anonymous {
         exchange_mode main;
         doi ipsec_doi;
         situation identity_only;
         generate_policy on;
         my_identifier asn1dn;
         peers_identifier asn1dn;
         verify_identifier on;
         certificate_type x509 "mycert" "mypriv";
         verify_cert off;
         proposal {
                 encryption_algorithm 3des;
                 hash_algorithm sha1;
                 authentication_method rsasig;
                 dh_group modp1024;
         }
}
sainfo anonymous {
         lifetime time 28800 sec;
         encryption_algorithm 3des ;
         authentication_algorithm hmac_md5;
         compression_algorithm deflate ;
}
```

For Openswan my racoon.conf looks like this:

```
path certificate "/etc/racoon/certs";
padding{
     maximum_length 20;     # maximum padding length.
     randomize off;          # enable randomize length.
     strict_check off;       # enable strict check.
     exclusive_tail off;     # extract last one octet.
}
# Specification of default various timer.
timer{     # These value can be changed per remote node.
   counter 5;              # maximum trying count to send.
   interval 20 sec;        # maximum interval to resend.
   persend 1;              # the number of packets per a send.
     # timer for waiting to complete each phase.
   phase1 30 sec;
   phase2 30 sec;
}
remote anonymous {
   exchange_mode main, aggressive;
   doi ipsec_doi;
   situation identity_only;
   certificate_type x509 "mycert" "mypriv";
   verify_cert off;
   my_identifier asn1dn;
   peers_identifier asn1dn;
   verify_identifier off;
   lifetime time 28800 seconds;
   initial_contact on;
   passive off;
   proposal_check obey;
   support_mip6 on;
   generate_policy off;
   nonce_size 16;
   proposal {
      encryption_algorithm 3des;
      hash_algorithm md5;
      authentication_method rsasig;
      dh_group modp1024;
   }
}
sainfo anonymous {
      lifetime time 28800 seconds;
      encryption_algorithm 3des, aes 128;
      authentication_algorithm hmac_md5;
      compression_algorithm deflate;
   }
   listen {

   }
   log debug;
```

Finally you copy your certificate pair (pem file and stripped rsa-key-file) to */etc/racoon/certs.*

A wonderful way to let you create config-files is a free tool called IPSecuritas, which uses Mac OS X's built in racoon, but uses a GUI. With this you can do some experimentation, and tweaking, and finally use the resulting config-file (they secretly place that config-file in /tmp/ while an IPsec-connection is running, and racoon is up)

2.4.3 Linux

Linux as a client is similar to Linux as a Server. I use racoon as a client, because configuring racoon is the same on Linux as it is on all BSDs. These instructions should be portable to BSD-OSes, which manage to get l2tpd to work.

The Kernel configuration has to meet the requirements in 2.3.2. Racoon and the ipsec-tools have to be installed. Furthermore l2tpd must be installed.

My racoon.conf on the client looks like the following:

```
path certificate "/etc/racoon/certs";
padding {
    maximum_length 20;
    randomize off;
    strict_check off;
    exclusive_tail off;
}
# Specification of default various timer.
timer {     # These value can be changed per remote node.
    counter 5;
    interval 20 sec;
    persend 1;
    phase1 30 sec;
    phase2 30 sec;
}
remote anonymous {
  exchange_mode main, aggressive;
  doi ipsec_doi;
  situation identity_only;
  certificate_type x509 "mycert" "mypriv";
  verify_cert off;
  my_identifier asn1dn;
  peers_identifier asn1dn;
  verify_identifier off;
  lifetime time 28800 seconds;
  initial_contact on;
```

```
    passive off;
    proposal_check obey;
    support_proxy on;
    generate_policy off;
    nonce_size 16;
    proposal {
      encryption_algorithm 3des;
      hash_algorithm md5;
      authentication_method rsasig;
      dh_group modp1024;
    }
  }
  sainfo anonymous {
    lifetime time 28800 seconds;
    encryption_algorithm 3des, aes 128;
    authentication_algorithm hmac_md5;
    compression_algorithm deflate;
  }
  listen {
    # isakmp ;
  }
  log notify;
```

My Policy for the connection looks like this:

```
spdadd <vpn-server>/32 <local>/32 any -P in ipsec esp/transport//require;
spdadd <local>/32 <vpn-server>/32 any -P out ipsec esp/transport//require;
```

Beyond that, l2tpd.conf looks like this:

```
[lns default]
exclusive = no
ip range = 192.168.0.1-192.168.0.20
local ip = 192.168.1.2
name = myhostname
ppp debug = yes
pppoptfile = /etc/l2tpd/otions.l2tpd
call rws = 10
tunnel rws = 4
flow bit = yes
[lac vpn]
lns = 192.168.0.1
pppoptfile = /etc/l2tpd/options.l2tpd
redial=yes
max redials = 5
ppp debug = yes
local ip = 192.168.0.100
remote ip = 192.168.0.1
```

And my options.l2tpd:

```
ipcp-accept-local
ipcp-accept-remote
user wogri
password password
noauth
crtscts
idle 1800
defaultroute
nodetach
nodeflate
nobsdcomp
debug
lock
connect-delay 5000
```

Connect to your VPN by telling l2tp to establish the connection:

```
echo “c vpn” > /var/run/l2tpd-control
```

Or use this script (which also adjusts routes):

```
#!/bin/bash
if [ -z $1 ]
then
        echo please specify an interface!
        exit
fi
IP=$(ifconfig $1 | grep "inet " | awk \
'{ print $2 }' | cut -f 2 -d ':')
ROUTE=$(ip route list | grep default | awk \
'{ print $3}')
setkey -c << EOF
flush;
spdflush;
spdadd 192.168.0.101/32 $IP/32 any -P in ipsec esp/transport//require;
spdadd $IP/32 192.168.0.1/32 any -P out ipsec esp/transport//require;
EOF
echo "c vpn" > /var/run/l2tp-control
sleep 10
route del -net default
route add -host 192.168.0.1 gw $ROUTE
route add -net default gw 192.168.0.1
```

The echo command will let L2TPD initiate a connection to the l2tpd on the server-side. The IPsec policy says, that IP traffic to this connection is encrypted, so the kernel will initiate an IPsec connection to the remote host, then l2tp shakes hands, and finally the ppp daemons shake hands and connects.

2.4.4 Pocket PC 2003

Unfortunately, Pocket PC 2003 does not want support .p12 certificate files. It prefers a Microsoft proprietary format. *Fortunately*, somebody reverse-engineered this format, and wrote a conversion tool to convert openssl certificates to this format. *Unfortunately*, Microsoft does not provide a utility to import certificates, it only provides a utility to delete certificates[6]. *Fortunately*, somebody wrote a tool to import those certificates into the handheld devices certificate store.

To create such a certificate, use these commands (the pvk utility can be found on http://www.jacco2.dds

```
openssl crl2pkcs7 -certfile newcert.pem -certfile \
./demoCA/cacert.pem -nocrl -outform PEM -out usercrt.p7b
pvk -in newreq.pem -topvk -nocrypt -out userkey.pvk
```

Next, get the crtimport-utility from http://www.jacco2.dds.nl/networking/crtimprt.html, and copy the cert, the certimport-utilty and the crtimport.cfg-file to the handheld device (Store all these files in 'My Documents', otherwise the import will fail, unless you change crtimport.cfg). Run the crtimprt executable, it should tell you, that the certificates have been imported fine. Set up the VPN-Connection with the tool, Microsoft provides, and you're done.

2.5 Firewall

All this configuration assumes, that the VPN-box 'sits' in the network segment, where all the other workstations reside. It is very important to firewall the VPN-box, as L2TPD would listen on port 1701, and an attacker could easily pass around the IPsec Security Layer, if this port was world-wide open. Only 2 rules need to be allowed to go to the VPN-box: Port 500 / UDP for ISAKMP and the ESP Protocol.

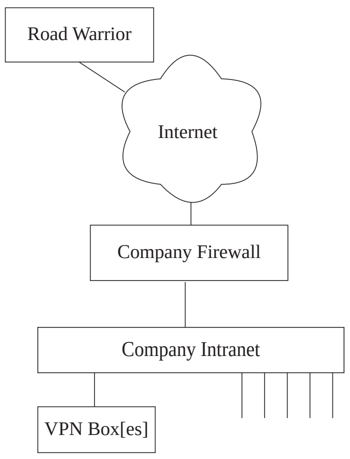
Given, the firewall is a stateful Linux-firewall, the rules would look like this:

```
iptables -A FORWARD -d $vpn -p esp -j ACCEPT
iptables -A FORWARD -d $vpn -p udp --dport 500 -j ACCEPT
```

2.6 Illustrations

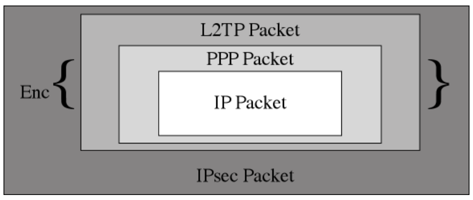
2.6.1 Network Diagram

Below the networking diagram, which assumes that the VPN-Box has an official IP-Address and the firewall routes this traffic to the VPN-Box.



[6]I think that is pretty funny; Well done, Microsoft.

2.6.2 How a packet is encapsulated



An outgoing IP packet is encapsulated in a PPP packet. This PPP packet is encapsulated in a L2TP packet. Again, this encapsulation does not make any sense, but it is declared 'Standard' from Microsoft. The resulting L2TP packet is finally encrypted (Enc { }) by the IPsec stack, the IPsec headers and trailers are added, and the resulting ESP packet leaves the network interface.

The receiver of the packet strips the packet using the decapsulation the other way round.

2.7 Security

2.7.1 General

The definition of the term security is very subjective. VPN-Connections are one of the greatest security-threats because road-warriors are not under the control of the sysadmins and usually 'invisible' to the network structure. If the laptop gets stolen, or just in the hands of a bad person, the intruder can do much harm to the network.

2.7.2 IPsec Protocol

The IPsec protocol itself is considered secure, there doesn't seem to be a better supported or more secure protocol that would meet all these requirements. As all the data which flows over the network is encrypted with strong algorithms, and the certificate method is considered secure, eavesdropping of the line should not easily be possible. Physical security is important, though, as the certificates are stored on the client machines. With the certificate in the bad man's hands, eavesdropping is definitely possible.

2.7.3 Network Layout

The VPN box sits on the internal network. There is no other place it should be put, unless the companies network infrastructure allows for that (DMZ for example). Some example implementations have dual-homed VPN-boxes. I consider that very insecure, as an intruder may bypass all firewall systems because he has control over the VPN box and can therefore hide traffic.

2.8 Security Improvements

2.8.1 Read-Only Filesystem / Boot from Compact Disc [Lnk: BCD]

The more secure, the better. CD-Rom drives are a very common device, and can be found in nearly every computer nowadays.

2.8.1.1 The Concept

Is it possible to boot a working operating system from a read-only disk? Yes it is, but one has to pay attention that some locations on a Unix-system should still be writeable. The following directories are writeable on my example box:

```
/var/log              Logfiles
/tmp                  Temporary Files
/var/run              Sockets and PID-Files
/var/lock             Lockfiles
/var/spool/postfix    Files for my MTA
```

Start out by copying your current installation to another partition, change the init-scripts to not remount the root-filesystem (and no other partition, if you use multiple partitions) in read-write mode, and modify or insert your own init-scripts to make the system boot into working[7] read-only mode.

2.8.1.2 Ramdisks

A ramdisk is a reserved space of memory, which can contain a filesystem, and thus, be mounted in the system. We use ramdisks for providing writeable space on our read-only filesystem. Every time, the computer is rebooted, the contents of the ramdisks are lost. If you want permanent writeable space, you must either use nfs (or another networking filesystem), or use a remote syslogd, if you only want permanent logs.

Format a ramdisk, which will provide 8 Megabyte of Space with the command:

```
mke2fs -q -I 1024 /dev/ram1 8192
```

You cam mount the ramdisk afterwards with the usual mount-command.

2.8.1.3 Random Access Memory Amount

Ramdisks use or waste (depends on your point of view) memory. As you don't have a hard-disk, you can not swap memory. Therefore it is important to equip your machine with enough Memory to bear the load.

My system uses approximately 40 megabytes of RAM after booting. If I initiate an IPsec connection to the box, it uses another 0.5 Megabytes of RAM, for each additional connection. The postfix daemon itself uses some memory during processing of e-mails (I use it for notification E-Mails, see 2.8.1.9). So 512 or 1024 megabytes of RAM should be enough for a production system with about 100 users.

2.8.1.4 Device File System

According to http://www.linuxjournal.com/article.php?sid=7233 it is recommended to use the devfs, the device file system. Why? This proc-like filesystem for device nodes also resides in RAM, and gives us more flexibility. What you will need:

- A devfs enabled kernel (see 2.8.1.7 for all kernel requirements), which mounts the devfs on boot.
- The devfsd, which starts before any other program called by init

[7]Working in this sense means: Providing a working VPN-Server.

What happens, is the following:

The devfsd finds all registered hardware, and creates symbolic links in the /dev/ directory. All the devices that really exist in the system are symlinked, and all device nodes are both read and writeable.

2.8.1.5 The /dev/pts directory

To be able to fork pty's, which is used for *ssh* and *VPN* sessions, you also need the /dev/pts filesystem in your kernel, which improves security quite a bit. It creates pty device nodes on the fly, with the rights of the actual user.

According to the description on the developer of the devfs, devfs should handle the pts directory. This didn't work for me, so I chose to use the /dev/pts filesystem, and things worked perfectly.

2.8.1.6 The Bootloader

To load a kernel from CD, you have to use a special bootloader. I chose the fedora core 2 boot-CD (CD1), which contains a folder called isolinux. Copy the contents of this folder to your hard-drive. Edit syslinux.cfg to fit your needs, and copy the kernel in that directory. That's it, the bootloader will do the rest for you. Deeper information on how to place to bootloader on it's right place can be found in 2.8.1.10.

2.8.1.7 The Kernel

in order to boot from the CD you need some minor kernel modifications. First of all, you have to support the filesystem you boot from (iso9660), and all the pseudo-filesystems (devfs, /dev/pts, proc at least). Of course you need ramdisk support.

```
CONFIG_PROC_FS=y
CONFIG_DEVFS_FS=y
CONFIG_DEVPTS_FS_XATTR=y
CONFIG_RAMFS=y
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_SIZE=8192
CONFIG_ISO9660_FS=y
CONFIG_JOLIET=y
```

2.8.1.8 Init-Scripts

You will need to do some init script modifications. The first init script that will be called on fedora systems is /etc/rc.d/rc.sysinit, which is where you have to comment out the remount for / to read-write mode, and insert the mounts for the ramdisks. You might want to disable some daemons, that you don't need, and make some changes in /etc/fstab, according to your configuration.

2.8.1.9 E-Mail Notifications

I chose postfix as an MTA, as mentioned above. I bound it only to the loopback-interface, so it doesn't listen on any physical network interface. I deactivated logrotation, and a shellscript greps the times and ip-addresses of connections, and sends it to the SA-department. Then I do the logrotation myself, with the few files that I really want to rotate.

My script looks like this:

```
#!/bin/bash
# report and rotate logs
echo "From: VPN-Master <root@company.net>" > \
/tmp/daily_report
echo "To: admin@company.net" >> \
/tmp/daily_report
echo "Subject: VPN Connections - Daily report"\
>> /tmp/daily_report
grep l2tpd /var/log/messages | grep established \
| grep Call | awk '{print ($1 " "  $2 "\
" $3 " " $10)}' | sed 's/,//' >> \
/tmp/daily_report
grep Call | cut -d ' ' -f 1-3,10 | sed 's/,//'\
>> /tmp/daily_report
/usr/sbin/sendmail admin@company.net < /tmp/daily_report
```

Logrotation (in this example only for one file, just to get the idea) is done by:

```
ln /var/log/messages /var/log/messages.0
rm /var/log/messages
killall -HUP syslogd
```

2.8.1.10 The ISO Filesystem

This is how to create the ISO-Filesystem for the CD. The trick is to tell mkisofs not to stick to the standards. Use long filenames, merge UNIX-permissions and owners into the filesystem, merge special characters - in short: do everything, that is not allowed by the standard. We don't care, if every operating system can read our CD, because the CD contains the operating system which definitely can read the CD itself. Read man mkisofs to get the whole detail about the switches.

```
mkisofs -l -U -R -b isolinux/isolinux.bin -c isolinux/boot.cat  \
-no-emul-boot -boot-load-size 4 -boot-info-table -o /mnt/to_burn/IPsec_vpn.iso .
```

The -b switch is the important one. It tells mkisofs to create a bootable CD. The switch *-no-emul-boot* defines the type of boot-cd. This mode will not emulate a floppy (as the 2.6 kernel without modules usually won't fit on a floppy), nor hard-disk mode. It will tell the bios to boot the bootloader (defined in the -b switch) from the CD.

Luckily, mkisofs adds all these command-line switches to the iso itself. A hexdump of the original fedora-cd shows the mkisofs command (within the first 100 lines).

2.8.2 Encrypted Certificate Authority

I like to have the CA on my CD, but I don't want to expose the CA directly to the filesystem. So I created a tar archive of the CA, and encrypted[8] it with a symmetric key. With shellscript, it will be decrypted, untarred and mounted on a ramdisk.

If you have your own dedicated CA somewhere else, there is no need to put it on the CD.

[8]

```
gpg --symmetric --cipher-algo 3DES --output ca.tar.bz2.gpg ca.tar.bz2
```

Chapter 3

Results

What are the results of my project?

- A VPN-Server that supports every common *Road Warrior* operating system
- No special clients needed for any operating system, just the standard, built-in clients
- Easy setup for the client side
- Encryption protocols that are standards based and known to be secure
- No license issues.
- No limits on supported clients, except for hardware limits
- No hardware dependencies, as everything boots of a CD. Broken Hardware is swapped out in a few minutes.
- Based on the stable Linux operating system
- Support for NAT-Traversal
- Support for Pocket PC 2003 (which I like to call the CEO's joy feature)

Bibliography

[Lnk: OSW] http://www.openswan.org

[Lnk: KME] http://www.kame.net/racoon

[Lnk: JDL] http://www.jacco2.dds.nl/networking/freeswan-l2tp.html

[Lnk: NCa] http://www.natecarlson.com/linux/ipsec-x509.php

[Lnk: BCD] http://www.linuxjournal.com/article/7233