Microsoft®

Windows Vista ™ Beta/Linux IPsec Interop Testing

Authors: Hank Janssen, Chris Rawlings, Sam Vaughan

Abstract:

This document provides an overview of Linux IPsec solutions as well as detailed discussions on configuring IPsec-Tools for interoperability scenarios between Red Hat Linux Enterprise 4 and Windows Vista Ultimate Beta.

Table of Contents

1 Introduction

1.1 Scope of Document and Testing

This document describes the configuration and testing of IPsec interoperability scenarios performed between Linux and Windows Vista at the Microsoft OSS Labs. It provides an overview of the available IPsec solutions in Linux, detailed walkthroughs on the configuration of IPsec-Tools in Linux, and the results and insights obtained from the testing.

The primary goal of the testing was to discover the level of IPsec interoperability between Linux and Microsoft's soon to be released Windows Vista operating system. Another goal of the project was to help improve Windows Vista IPsec functionality by identifying and fixing bugs as they were encountered during the testing process.

The testing for this project fell within the following scope:

- IPv4 - All IPsec testing was done using IPv4
- IPsec-Tools - All IPsec testing on Linux was done using the IPsec-Tools package (version 0.3.3). If needed, additional testing was done using a manually compiled version (0.6.6).
- All IPsec configurations on Windows Vista were setup using the new Advanced Firewall and Security snap-in (MIPS). Regression testing was done using the legacy IPsec Policy snap-in (LIPS) if a bug was found with MIPS.

Within this scope, the following scenarios are among those that were tested:

- Authentication Testing - Pre-Shared Keys (PSK's), Certificates
- IPsec in Transport Mode
- IPsec in Tunnel Mode
- Lifetimes and Rekeying
- NAT-T - NAT-Traversal for IPsec connections
- Security Methods - AES, 3DES, etc and Multiple Security Method testing
- Traffic Over SA's - Sending real-world traffic over established IPsec connections.

In addition to providing a record of the interoperability testing performed, a major focus of this document is to provide a resource to Windows IPsec experts that will allow them to successfully configure an IPsec connection in Linux using IPsec-Tools.

1.2 Out of Scope

The following items are NOT within the scope of this document or the testing that was performed:

- Windows Vista configuration - In general, this document does not provide information on configuring IPsec in Windows Vista. It only provides configuration examples and instruction for Linux.
- Exhaustive configuration list - This document does not contain example configurations or results for every single test scenario. A large effort has been made, though, to include examples for most major configuration changes.
- IPsec over IPv6 - No IPsec testing was done using IPv6.
- IPsec and Kerberos - No IPsec testing was done using Kerberos authentication.
- Other 3rd Party Software/Hardware IPsec Solutions - KLIPS, OpenSwan, StrongSwan, FreeSwan, and other IPsec solutions were not tested. IPsec-Tools was the only IPsec solution tested.

2 Linux IPsec Overview

2.1 Available IPsec Implementations

Before diving into the IPsec configuration details of the interoperability testing, it is important to understand the various IPsec implementations available on Linux. Unlike Windows Vista, there are multiple implementations and tools to choose from. There are two components in Linux that are needed to setup a working IPsec connection: a kernel IPsec implementation and a user-space package that includes an IKE daemon.

Kernel IPsec Implementations
A native IPsec stack, known as NETKEY or 26sec, has existed in the Linux kernel since version 2.5.47. There is also another IPsec kernel implementation known as KLIPS that also exists, but this document will only cover NETKEY since it is the standard for 2.6 kernels.

User-Space Tools/Software
There are several options available, as far as user-space tools are concerned, to manage and configure the IPsec features provided in the kernel by NETKEY. Each competing package includes an Internet Key Exchange (IKE) server as well as tools for configuring an IPsec connection:

- IPsec-Tools – This is the default package installed with Red Hat Enterprise Linux 4 (RHEL4) and Fedora Core 5 (FC5).
- Openswan – A good alternative to IPsec-Tools that is in active development. It seems to be most popular in VPN scenarios.
- strongSwan – Another alternative to IPsec-Tools, but not quite as common as Openswan.
- FreeS/WAN – One of the first IPsec packages available on Linux which doesn't see much development anymore.
- ISAKMPD – This is a Linux port of the IPsec software available on OpenBSD

All of these competing packages can be used with NETKEY, but IPsec-Tools is the only one that is pre-installed on RHEL4, FC5, and most of the other distributions such as Ubuntu. Since it is the default IPsec package available and will likely see accelerated use and continued development in the future, it was chosen along with NETKEY for all of the testing in this document.

NOTE: In contrast to Windows Vista, IPsec in Linux is not closely tied to the firewall. The use of a firewall is optional and it is configured separately from IPsec-Tools using iptables (the name of the Linux firewall software).

2.2 IPsec-Tools

The two IPsec components chosen for our testing were NETKEY and IPsec-Tools, as mentioned above. NETKEY provides the core features and network functionality required for IPsec, while IPsec-Tools provides an IKE daemon and utilities to configure various IPsec policies and options.

The following is a list of the main command-line tools included with IPsec-Tools:

- *setkey* - Used to configure Security Policies (SP's) for an IPsec connection and manually configure Security Associations (SA's) if needed.
- *racoon* - The IKE daemon used for IPsec negotiations.
- *racoonctl* - An administrative control tool for racoon. This was not needed for our testing and therefore will not be covered in this document.

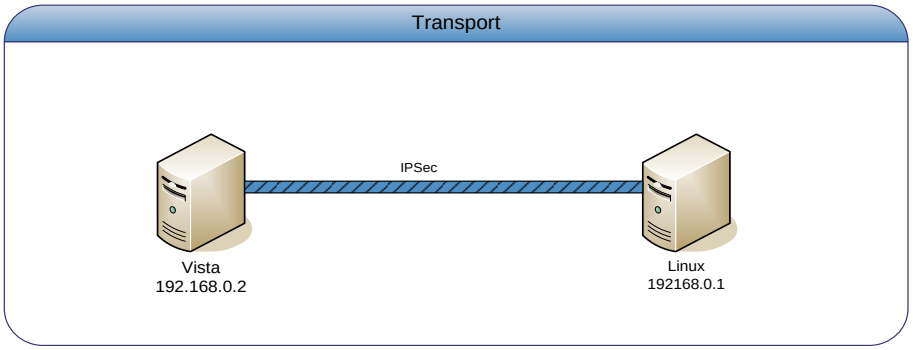
The two main configuration files required for most IPsec configurations are listed below:

- setkey.conf - Used to store settings for setkey and typically located in /etc/racoon.
- racoon.conf - Used to store settings for racoon and typically located in /etc/racoon. racoon.conf can also reference a psk.txt file when using pre-shared keys and/or a certificate directory when authenticating with certificates.

2.2.1 IPsec-Tools Configuration (setkey.conf)

The following two sections provide an introduction to the setkey.conf file and describe the differences in configuration when using transport mode or tunnel mode. Almost all of the testing performed was done in end-to-end transport mode, but there were also a few tests done in tunnel mode.

2.2.1.1 Transport Mode



The following setkey.conf file would be used for the host-to-host transport scenario shown above. The setkey.conf file shown is for the host with the IP address 192.168.0.2. The setkey.conf file for the other host (192.168.0.1) would be the same except that the inbound and outbound policies would be switched with each other (i.e. you would need to switch the 'in' and 'out' keywords in the two policy rules).

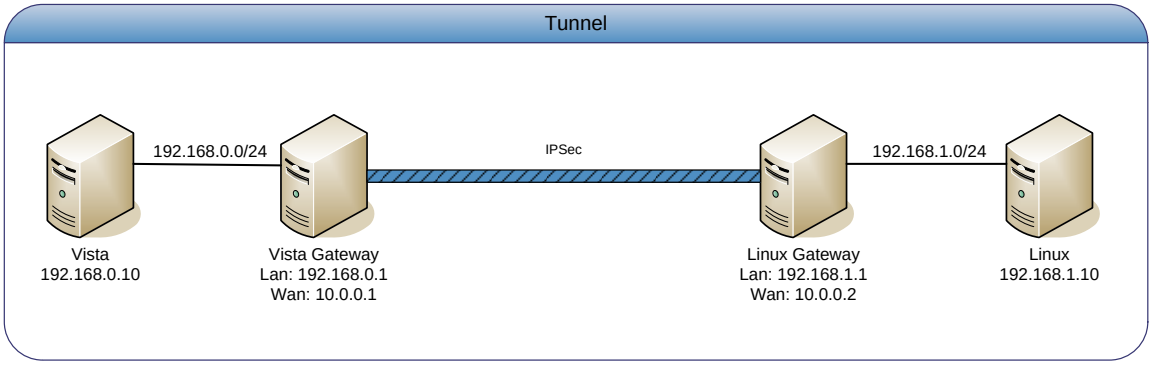
setkey.conf (transport mode)

```
# Security Policies

spdadd 192.168.0.2 192.168.0.1 any -P out ipsec
       esp/transport//require;
spdadd 192.168.0.1 192.168.0.2 any -P in ipsec
       esp/transport//require;
```

Since IPsec policies are unidirectional, there are two security policy lines that need to be configured. Each security policy in the configuration file begins with the 'spdadd' keyword and ends with a semi-colon so the security rule can span multiple lines. In this example, the first security policy rule can be read to say 'Require all outbound traffic (matching any/all protocols) from 192.168.0.2 to 192.168.0.1 to be secured with IPsec using ESP in transport mode. The second security policy rule is simply a mirror of the first rule for inbound network traffic. The inbound and outbound rules do not have to match, so you could apply different settings to one direction of traffic if desired.

2.2.1.2 Tunnel Mode



For the gateway-to-gateway tunnel scenario shown above, the Linux gateway would need the following setkey.conf. In this example, there is an IPsec tunnel between the Windows Vista and Linux gateway machines. The gateways are configured to secure all network traffic between the two subnets.

setkey.conf (tunnel mode)

```
# Security Policies

spdadd 192.168.1.0/24 192.168.0.0/24 any -P out ipsec
       esp/tunnel/10.0.0.2-10.0.0.1/require;
spdadd 192.168.0.0/24 192.168.1.0/24 any -P in ipsec
       esp/tunnel/10.0.0.1-10.0.0.2/require;
```

The configuration file is very similar to the one used for transport mode connections. As shown, setkey.conf is configured to protect all network traffic between the two subnets (192.168.0.0/24 and 192.168.1.0/24) using a tunnel between the two gateway IP addresses (10.0.0.1 and 10.0.0.2). Once again, the second policy rule is a mirror of the first.

NOTE: As with Windows Vista, there are not any special routes in Linux that need to be manually configured for the network traffic between the two subnets. The routes are taken care of by IPsec automatically and will not show up in the routing table.

2.2.2 IPsec-Tools Configuration (racoon.conf)

This section provides an introduction to the racoon configuration file, racoon.conf. A simple racoon.conf example is given below along with an explanation of each option used.

racoon.conf

```
# Racoon IKE daemon configuration file.
# See 'man racoon.conf' for a description of the format and entries.

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

# Phase 1 (Main Mode) Configuration
remote anonymous
{
        #IKE Exchange Mode
        exchange_mode main;

        # Phase 1 (Main Mode) Proposal
        proposal {
                authentication_method pre_shared_key ;
                encryption_algorithm 3des ;
                hash_algorithm sha1 ;
                dh_group 14 ;
        }
}

# Phase 2 (Quick Mode) Configuration/Proposal (for IPsec SA).
sainfo anonymous
{
        encryption_algorithm 3des ;
        authentication_algorithm hmac_sha1 ;
        compression_algorithm deflate ;
}
```

There are two main sections in the configuration file that correlate to phase 1 and phase 2 of an IKE negotiation:

- **remote anonymous {…}** - This section defines the parameters used during phase 1 (main mode). The 'remote' keyword can be followed by either 'anonymous' or an ip address. If an IP address is specified, the settings will only be used for that IP. You can define multiple 'remote' sections, but only one of them can use the 'anonymous' keyword. The anonymous settings are used by racoon for any host whose IP address does not match another 'remote' section.
- **sainfo anonymous {…}** - This section defines the parameters used during phase 2 (quick mode). As with the 'remote' section, a specific IP address can be used instead of anonymous (but only one anonymous section can be defined).

Here is a more detailed explanation of some of the configuration options seen in racoon.conf:

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

  This path option tells racoon where to find the text file containing a list of pre-shared keys. There is also an additional path option that tells racoon where the certificate directory is located. Refer to racoon.conf in the certificate section of this document for an example.

Phase 1/ Main Mode Options (remote)

- `exchange_mode main ;`

  This option tells racoon what exchange mode to use for phase 1 when racoon is initiating and it also serves as a list of acceptable modes when racoon is responding. In addition to 'main' mode, it is possible to specify 'aggressive' and 'base'. The aggressive and base modes were not used for any of our testing.

- `proposal {…}`

  This sub-section defines a set of authentication and encryption options that may be used for phase 1. You can define multiple authentication methods for phase 1 by creating multiple 'proposal' blocks within the phase 1 section (i.e. the 'remote' section).

- `authentication_method pre_shared_key ;`

  This tells racoon to use pre-shared key authentication. When certificates are being used this line should be changed to '`authentication_method rsasig ;`'

- `encryption_algorithm 3des ;`

  The available encryption algorithms values for phase 1 are des, 3des, blowfish, cast128, and aes. When using aes, a key length needs to be specified (e.g. '`encryption_algorithm aes 128 ;`'). Our testing showed that key length values of 128, 192, and 256 worked correctly. The only algorithms common to both Racoon and Windows Vista are des, 3des, and aes.

- `hash_algorithm sha1 ;`

  The available authentication algorithm values for phase 1 are md5, sha1, sha256, sha384, and sha512. The only algorithms common to both Racoon and Windows Vista are md5 and sha1.

- `dh_group 14 ;`

  This specifies the diffie-hellman group with possible values of 1, 2, 5, 14, 15, 16, 17, or 18. These groups can be alternatively specified as modp768, modp1024, modp1536, modp2048, modp3072, modp4096, modp6144, or modp8192.

  NOTE: Racoon does not support the elliptic curve diffie-hellman groups that are new to Windows Vista. The only DH groups that are common to both Racoon and Windows Vista are 1, 2, and 14.

Phase 2 / Quick Mode Options (sainfo)

- `encryption_algorithm 3des ;`

  The list of available encryption algorithms for phase 2 is different than the for phase 1. The possible phase 2 encryption values are des, 3des, des_iv64, des_iv32, rc5, rc4, idea, 3idea, cast128, blowfish, null_enc, twofish, rijndael, and aes. The only algorithms common to both Racoon and Windows Vista are des, 3des, and aes.

- `authentication_algorithm hmac_sha1 ;`

  The list of available authentication algorithms for phase 2 includes des, 3des, des_iv64, des_iv32, hmac_md5, hmac_sha1, hmac_sha256, and hmac_sha384. The only algorithms common to both Racoon and Windows Vista are hmac_md5 and hmac_sha1.

- `compression_algorithm deflate ;`

  This option is used for IPComp.

There are many additional settings that can be configured in addition to what is seen in the example configuration above. Refer to the racoon.conf man page for an exhaustive list of options:

```
[root@host]# man racoon.conf
```

2.2.3 IPsec-Tools Administration

The following command line directives show how to start an IPsec connection once setkey.conf and racoon.conf have been configured correctly. These steps are required to start/stop all IPsec connections

using IPsec-Tools. Also, all of the steps assume that the configuration files are stored in /etc/racoon (we used /etc/racoon, but the configuration files can be stored anywhere).

2.2.3.1 Starting IPsec

First, the desired security policies need to be loaded into the kernel from setkey.conf. With a correctly configured setkey.conf file, the following setkey command will return without any errors:

```
[root@host]# setkey -f /etc/racoon/setkey.conf
[root@host]#
```

The setkey command is used to manually manipulate the SA and SP databases and in this case we tell it to load settings from /etc/racoon/setkey.conf.

Next, the IKE daemon needs to be started by issuing the following command:

```
[root@host]# racoon -f /etc/racoon/racoon.conf -v -ddd -l /etc/racoon/racoon.log
[root@host]#
```

The racoon command above tells the racoon daemon where to find the configuration file and specifies a file for logging messages in verbose debugging mode.

Finally, to establish the IPsec connection, ping the remote host defined in the security policy (in setkey.conf). For this simple example, we will pretend the remote host is 192.168.1.1. The first ping attempt will show the 'temporarily unavailable' message while the IPsec connection is being established. When it is established, a second ping attempt will work.

```
[root@host]# ping 192.168.1.1
connect: Resource temporarily unavailable
[root@host]# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=0 ttl=127 time=0.350 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=127 time=0.262 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=127 time=0.312 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.262/0.308/0.350/0.036 ms, pipe 2
[root@host]#
```

Please refer to the troubleshooting section of this document for information on determining whether or not network traffic is actually being protected by IPsec.

2.2.3.2 Stopping IPsec

In order to stop IPsec, the racoon IKE daemon needs to be 'killed' with the following command:

```
[root@host]# killall racoon
[root@host]#
```

Then issue the following two commands to flush the SA's and SP's from the kernel:

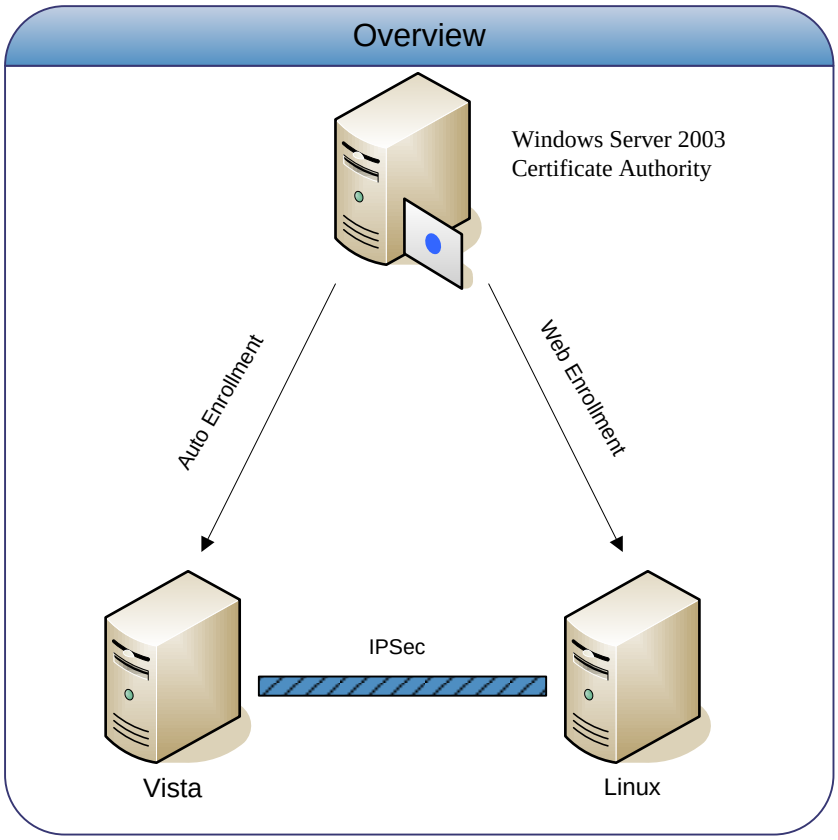
```
[root@host]# setkey -F
[root@host]# setkey -FP
```

Optionally, you can also double check that the SA and SP settings have been flushed by issuing the following commands:

```
[root@host]# setkey -D
No SAD entries.
[root@host]# setkey -DP
No SPD entries.
```

3 Testing Environment

3.1 Overview Diagram



3.2 Hardware

The following server and desktop hardware was used. All servers and desktops were hosted in the Microsoft OSS Lab.

Machine Role	OS	Hardware
Certificate Server	Windows Server 2003 R2 Enterprise Edition	Compaq Blade BL10e
Linux IPsec Host	Fedora Core 5	PC Workstation
NAT device	N/A	DLink SI-624S
NAT device	N/A	Netgear RO318
Linux IPsec Host	Red Hat Enterprise Linux 4 AS	Dell Blade PowerEdge 1855
Linux IPsec Host	Red Hat Enterprise Linux 4 AS	Dell Blade PowerEdge 1855
Linux IPsec Host	Red Hat Enterprise Linux 4 AS	HP Proliant DL385
Linux IPsec Host	Red Hat Enterprise Linux 4 AS	HP Proliant DL385
Windows IPsec Host	Windows Vista Ultimate Edition Beta	DELL Optiplex GX260

Machine Role	OS	Hardware
Windows IPsec Host	Windows Vista Ultimate Edition Beta	DELL Optiplex GX 280
Windows IPsec Host	Windows Vista Ultimate Edition Beta	Dell Blade PowerEdge 1855
Windows IPsec Host	Windows Vista Ultimate Edition Beta	Dell Blade PowerEdge 1855
Windows IPsec Host	Windows Server 2003 R2 Enterprise Edition	Compaq Blade BL10e
Windows IPsec Host	Windows Server 2003 R2 Enterprise Edition	Dell Blade PowerEdge 1855
Windows IPsec Host	Windows XP Professional Edition	Compaq Blade BL10e
Windows IPsec Host	Windows XP Professional Edition	Compaq Blade BL10e

3.3 Software

3.3.1 Red Hat Enterprise Linux 4

Redhat Enterprise Linux 4 Advanced Server (RHEL4) was chosen as the distribution for testing the IPsec scenarios. All RHEL4 machines were default installations updated to the latest release/patches via up2date, Red Hat's system for managing and updating RPM packages for Red Hat Enterprise Linux. After running up2date the package for IPsec-Tools was updated to ipsec-tools-0.3.3-6.rhel4.1 which was the base version used for all of our testing.

The following is a list of additional software packages installed using up2date:

- kernel-devel (2.6.9-34.0.2.EL)
- kernel-smp-devel (2.6.9-34.0.2.EL)
- openssl-devel (0.9.7a-43.8)
- flex (2.5.4a-33)
- byacc (1.9-28)
- ethereal (0.99.0-EL4)

Software installed from source:

- IPsec-Tools-0.6.6 from http://ipsec-tools.sourceforge.net/

3.3.2 Windows Vista Ultimate Beta

The most recent versions of Windows Vista Ultimate Beta (available internally at Microsoft) were installed at the time of testing. When needed, the machines were periodically updated to the most recent builds and private builds that were provided by the IPsec team were also used to test specific bugs and fixes.

Additional software used during testing included:

- Putty
- Pscp
- Wireshark
- Netmon
- sfpcopy
- tracepdb
- tracelog
- tracefmt

3.3.3 Windows Server 2003 R2 Enterprise Edition / Windows XP Professional Edition

Windows Server 2003 R2 Enterprise Edition and Windows XP Professional Edition installations were default installs with all updates applied on 7/17/06 via Windows update.

Additional software used during testing included:

- Putty
- Pscp
- Ethereal/Wireshark
- Netmon

3.3.4 Certificate Server

The Windows Server 2003 certificate server also had the following server roles installed:

- Application Server (for web enrollment)
- Domain Controller
- DNS Server
- Certificate Services

4 Authentication Testing Scenarios and Configuration

4.1 Test Descriptions

All of our authentication testing for this project focused on using the following authentication methods to establish an IPsec connection:

- PSK's (plain ASCII and Unicode keys)
- Certificates
- Multiple Authentication Methods

4.2 Test Scenarios / Setup

4.2.1 Pre-Shared Keys (PSK's)

To setup racoon to authenticate using Pre-Shared Keys two files need to be edited: racoon.conf and psk.txt. In racoon.conf make sure that the path to psk.txt is specified.

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

In the racoon.conf Phase 1 proposal section, the authentication method needs to be '`pre_shared_key`'.

```
proposal {
        authentication_method pre_shared_key ;
        encryption_algorithm 3des ;
        hash_algorithm sha1 ;
        dh_group 14 ;
}
```

When configuring pre-shared keys, each key entry in the psk.txt file should be in the format of:

`[identifier] [key]`

A '#' symbol prefixes a comment line. The identifier can be an IPv4 or IPv6 address, USER_FQDN, or FQDN. The Key is your secret key. When using Unicode keys make sure your terminal and text editor support UTF-8. The following is an example PSK configuration file (psk.txt):

```
# file for pre-shared keys used for IKE authentication

# Vista Testing ASCII
192.168.0.1 SuperSecretPassword

# Vista Testing Unicode
192.168.0.3 абвгдеёжзийклмно
```

4.2.2 Certificates

Using certificates instead of PSK's requires a little more setup in Linux, especially when the certificates are being issued by a Windows certificate server. In our test environment we used a Windows Server 2003 R2 Certificate Server to issue certificates to Windows Vista and Linux clients. Windows Vista clients received certificates through auto enrollment and Linux clients received certificates through the Web Enrollment

feature on the server. The following list shows the steps we used to generate working certificates for Linux:

1. Generate a certificate request in Linux
2. Retrieve a certificate from the server using Web Enrollment
3. Convert the issued certificate into the format required by racoon
4. Make the root certificate trusted on Linux
5. Make sure the issued certificate can be verified

These steps are explained in more detail below and followed up by an example on how to configure IPsec-Tools when using certificates.

4.2.2.1 Generating Certificates for Linux

1. Generate a Certificate Request in Linux

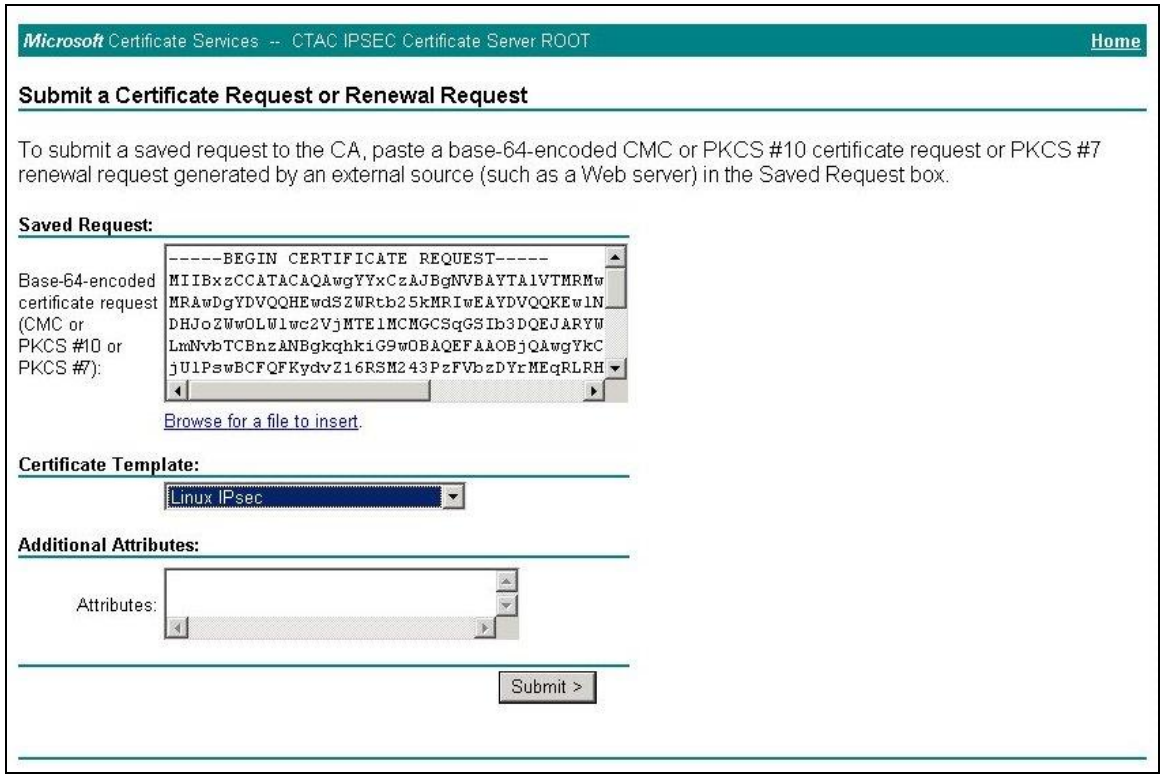
Create the certificate request using the openssl command below (openssl may need to be installed first). The output files, example.key and example.csr, can be renamed as needed. After the RSA key pair is generated you will be asked to enter some information that will be included with the certificate (e.g. country, state, email, server name).

```
[root@host]# cd /etc/racoon/certs
[root@host]# openssl req -new -nodes -keyout example.key -out example.csr
```

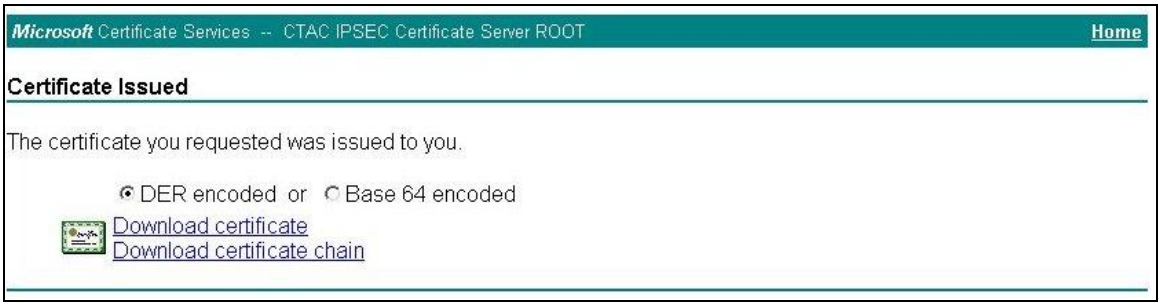
NOTE: If the /etc/racoon/certs directory does not exist, create it using 'mkdir /etc/racoon/certs'.

2. Retrieve a certificate from the server using Web Enrollment

Next, paste the entire contents of the CSR file (example.csr) into the Web Enrollment form on the certificate server as shown in the following image:



After the certificate has been issued, be sure to download it as a DER encoded file:



3. Convert the Issued Certificate and the Root CA Certificate Into the Format Required by Racoon

Copy the DER certificate (example.cer) that was issued by the certificate authority back into /etc/racoon/certs on the Linux client. Also, export the CA's root certificate and copy it into the same directory. Then convert both certificates into the PEM format required by racoon:

```
[root@host]# cd /etc/racoon/certs
[root@host]# openssl x509 -in example.cer -inform DER -out example.pem -outform PEM
[root@host]# openssl x509 -in rootca.cer -inform DER -out rootca.pem -outform PEM
```

4. Make the Root Certificate Trusted on Linux

There is an additional step needed in Linux to make the root certificate (rootca.pem) trusted so that the machine certificate (example.pem) can be verified. Create a symbolic link in /etc/racoon/certs using a hash feature of openssl:

```
[root@host]# cd /etc/racoon/certs
[root@host]# ln -s rootca.pem `openssl x509 -noout -hash -in rootca.pem`.0
```

IMPORTANT: Pay special attention to the back ticks (`). These are NOT apostraphes (').

6. Make Sure the Machine Certificate Can Be Verified

First, check the directory listing for /etc/racoon/certs to make sure everything was done correctly in the previous steps. You should see, at least, the following files:

```
[root@host]# cd /etc/racoon/certs
[root@host]# ls -l
lrwxrwxrwx  1 root root   16 Aug  2 14:51 a0ad47ad.0 -> rootca.pem
-rw-r--r--  1 root root  891 Jul 14 15:30 example.key
-rw-r--r--  1 root root 2451 Jul 14 15:58 example.pem
-rw-r--r--  1 root root 1866 Jul 14 15:59 rootca.pem
```

NOTE: After making sure the machine certificate can be verified using the next step below, you can delete any of the other intermediate files that were created during the certificate creation/conversion process (i.e. you can delete the .cer and .csr files).

If you can issue the following command successfully to verify the machine certificate, you know you have done things right:

```
[root@host]# cd /etc/racoon/certs
[root@host]# openssl verify -CApath /etc/racoon/certs/ example.pem
example.pem: OK
```

4.2.2.2 Certificate Policy Configuration

Once the certificates have been setup, there are only a few changes needed in racoon.conf to use certificates instead of PSK's:

- The path needs to be changed to a certificate path instead of a PSK path
- The my_identifier and certificate_type directives need to be added to the phase 1 settings
- The authentication method in the phase 1 proposal needs to be changed from 'pre_shared_key' to 'rsasig'

These changes are highlighted in the following racoon.conf example:

```
# Racoon IKE daemon configuration file.
# See 'man racoon.conf' for a description of the format and entries.

path certificate "/etc/racoon/certs" ;

# Phase 1 (Main Mode) Configuration
remote anonymous
{
        #IKE Exchange Mode
        exchange_mode main ;

        my_identifier asn1dn ;
        certificate_type x509 "yourcert.pem" "yourcert.key" ;

        # Phase 1 (Main Mode) Proposal
        proposal {
                authentication_method rsasig ;
                encryption_algorithm 3des ;
                hash_algorithm sha1 ;
                dh_group 14 ;
        }
}

# Phase 2 (Quick Mode) Configuration/Proposal (for IPsec SA).
sainfo anonymous
{
        encryption_algorithm 3des ;
        authentication_algorithm hmac_sha1 ;
        compression_algorithm deflate ;
}
```

4.3 Test Results

4.3.1 Known Issues

Two issues were discovered with Windows Vista during the authentication testing which affected several tests. The first issue involved Windows Vista sending a Group description attribute even when PFS was disabled on the Linux side. This caused the proposals to be rejected and no SA's to be established. The second issue dealt with PSK's when they were configured using MIPS. MIPS was saving PSK's in UTF-16 format which is incompatible with UTF-8, the standard for Linux. This effected interoperability with both ASCII and Unicode keys. NOTE: *Both of these issues have been fixed in Windows Vista.*

There were also two issues discovered with racoon and how it handles certificates. First, racoon correctly handles certificates issued by a single CA, but it is not able to use or verify certificate chains with any number of intermediate CA's. Second, it is not possible to specify multiple certificates in racoon like Windows Vista, so authentication tests involving multiple certificates could not be tested.

5 Main Mode Testing Scenarios and Configuration

5.1 Test Descriptions

The Main Mode (MM) tests included the following scenarios:

- MM Security Methods
- Multiple MM Security Methods
- MM Lifetime and Rekeying
- NAT-T

As with previous tests, we verified that the MM SA was negotiated successfully with the right parameters.

5.2 Test Scenarios / Setup

5.2.1 Lifetime and Rekeying

Defining a main mode lifetime requires only minor configuration changes. The following racoon.conf excerpt shows how a lifetime setting can be added to the 'remote' (phase 1) section:

```
…
remote anonymous
{
        #IKE Exchange Mode
        exchange_mode main ;

        lifetime time 20 min ;

        my_identifier asn1dn ;
        certificate_type x509 "yourcert.pem" "yourcert.key" ;
…
```

Lifetimes can be specified as seconds, minutes, or hours and there is not a minimum lifetime value that is enforced. It is possible, for example, to specify a one second lifetime even though it would not be practical to do so.

NOTE: The lifetime can only be configured as a time lifetime. Byte lifetimes used to be supported in racoon, but they are currently deprecated.

5.3 Test Results

5.3.1 Known Issues

NAT-T between Windows Vista and Linux is unsupported. This is due to Windows Vista only supporting NAT-T in transport mode and Linux only having support for NAT-T in tunnel mode making the two incompatible at this time. Work for NAT-T in transport mode for Linux is on the racoon developer's TODO list but at the time of this writing they are not actively working on it. Adding transport NAT-T would require additional code in the Linux kernel as well as IPsec-Tools.

There is a main mode rekeying issue with racoon that causes a break in connectivity for a certain period of time after a rekey. From our testing, this occurs when the main mode lifetime is set to a lower value than the quick mode lifetime. For example, when Windows Vista and Linux had the following settings, there would be a break in connectivity for about one minute and then the IPsec connection would work again:

Windows Vista: MM Lifetime (1 min), QM Lifetime (5 min)
Linux: MM Lifetime (1 min), QM Lifetime (6 min)

6 Quick Mode Testing Scenarios and Configuration

6.1 Test Descriptions

The Quick Mode (QM) tests included the following scenarios:

- QM Security Methods
- Multiple QM Security Methods
- QM Perfect Forward Secrecy (PFS)
- QM Lifetime and Rekeying
- AH + ESP
- Filters - Me to Any, Subnet to Subnet, Subnet to Any, etc.
- NAT-T

6.2 Test Scenarios / Setup

6.2.1 AH+ESP

The configuration of AH + ESP is done through setkey.conf:

```
# RHEL4-IPSEC1 <> VISTA
spdadd 192.168.0.1/32 192.168.0.2/32 any -P out ipsec
       esp/transport//require
       ah/transport//require;
spdadd 192.168.0.2/32 192.168.0.1/32 any -P in ipsec
       esp/transport//require
       ah/transport//require;
```

The order in which AH and ESP are applied at the packet level is depended on their order specified in the configuration file. The order in the configuration file is very important.

6.2.2 Filters

Filters are configured in the setkey.conf file. The following is an example of a subnet to subnet filter in setkey.conf:

```
spdadd 192.168.0.0/24[any] 192.168.0.0/24[any] any -P out ipsec
       esp/transport//require;
spdadd 192.168.0.0/24[any] 192.168.0.0/24[any] any -P in ipsec
       esp/transport//require;
```

The source and destination ranges conform to one of the following formats:

- address
- address/prefixlen
- address[port]
- address/prefixlen[port]

6.3 Test Results

6.3.1 Variable Key Length Issue

One variable key length issue was discovered while testing QM security methods. When a variable key length is used, such as AES and Windows Vista is the initiator, Linux will use the initiator's (peer's) key

length even if it is smaller. This is considered a bug in racoon and it is still unresolved. The following racoon.conf excerpt shows the warning that is produced as a result of this issue:

```
2006-07-28 13:46:07: DEBUG: my single bundle:
2006-07-28 13:46:07: DEBUG:  (proto_id=ESP spisize=4 spi=00000000 spi_p=00000000
encmode=Transport reqid=0:0)
2006-07-28 13:46:07: DEBUG:   (trns_id=AES encklen=128 authtype=hmac-sha)
2006-07-28 13:46:07: DEBUG:   (trns_id=3DES encklen=0 authtype=hmac-sha)
2006-07-28 13:46:07: DEBUG:   (trns_id=TWOFISH encklen=128 authtype=hmac-sha)
2006-07-28 13:46:07: WARNING: less key length proposed, mine:128 peer:256.  Use
initiaotr's one.
```

6.3.2 Multiple Transform Issue + Order of Transform

Two issues were discovered when testing AH+ESP.

1. Windows Vista was not handling multiple transforms per proposal, which is valid per RFC's. This bug has been fixed and Windows Vista now supports multiple transforms in a proposal.

2. The second issue was discovered while testing the fix for the first issue. Linux seems to be dependant on the order of transforms in the SA proposal. The RFC's do not specify an order so this is considered a Racoon issue and is still unresolved at the time of this writing.

6.3.3 Filter Issues

Two issues were discovered with Filters:

1. *Subnet to Subnet – Linux Initiator*

   The first is when Linux is the initiator. Subnet to Subnet fails to negotiate SA's. When Windows Vista is the initiator the SA's establish with no problems. The exact cause of this issue is unkown and is still under investigation.

2. *General to Specific - Windows Vista Initiator*

   With this scenario the SA's are established but this is opposite of the expected results. This filter test was expected to fail (e.g. the SA's were suppose to fail negotiation), but the SA's were actually established. The cause of this issue is currently unknown.

6.3.4 Unsupported Test Issues

There were several planned test cases that could not be completed because Racoon does not support byte lifetimes, only time lifetimes.

7 Traffic Over SA Testing Scenarios and Configuration

7.1 Test Descriptions

The 'Traffic Over SA' tests were used to verify traffic flows in both directions over an SA after it was established. These tests were done using 50 MB, 200 MB and 500 MB file transfers over the SA's. SSH, HTTP, and SMB traffic were used for testing. These tests included the following scenarios:

- IPsec Modes (end to end transport, gateway to gateway tunnel, etc)
- IPsec Protocol and Security Methods - These were a mixture of previous tests except that they were tested with large data transfers.

7.2 Test Scenarios / Setup

These tests used existing configurations with the exception of tunnel Gateway to Gateway.

7.2.1 Gateway to Gateway Setup

Most of the testing for this project used IPsec in transport mode, but there were also a few tests that required tunnel mode settings. When using tunnel mode, there are some minor changes that need to be made to setkey.conf on the Linux gateway in a gateway-to-gateway scenario. Refer to the tunnel mode section of this document for an example of a setkey.conf file.

In addition to modifying setkey.conf, the Linux gateway needs to have IP forwarding enabled. This can be accomplished with the following command if IP forwarding has been enabled in the kernel:

```
[root@host]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

7.3 Test Results

All of the tests in this section failed due to a bug in either the Linux kernel, racoon, or Windows Vista. These issues are explained in further detail below.

7.3.1 Traffic Over SA Issue (Linux/Racoon Issue)

All data transfers stall after some time between Linux and Windows Vista in our tests. This is possibly due to a QM rekey or PMTU related problem. This bug could be in Racoon or the Linux Kernel.

While we have not determined the exact cause of the problem, we have come across the following behavior. The following generally happens when the data is a 'pull' from the Linux client. This list is not exhaustive:

- Racoon/Netkey fails after lifetime (byte or time) limit has been reached. New SA's are generated, though traffic still uses the old SA's
- Racoon/Netkey fails after lifetime (byte or time) limit has been reached. New SA's are generated and used, but traffic still stops.
- A QM rekey stalls the data transfer. The rekey this happens on seems to be random. We have had successful data transfers of 200M-500M with multiple QM rekeys.
- Failed SA Negotiation after 2-5 rekeys and Racoon fails to respond to new negotiation from Windows Vista.
- Received a PMTU error from the Linux kernel when the data transfer stalls.
    - If we adjusted the MTU manually, up or down with ifconfig, immediately after the data transfer stalled, it would resume the transfer (otherwise the transfer would fail or have to be cancelled). The timing required to adjust the MTU and jump start the data transfer was very

precise, so this workaround did not always work (not to mention this is not a practical solution to the problem).

- Racoon 0.3.3 will allow a restart of the data transfer (IE stop the file transfer and then restart) Racoon 0.6.6 will not.
- We have tested using SCP, HTTP, and SMB traffic
- When testing with two VPC's (Virtual PC Images), we could not reproduce the problem. All data transfers succeeded.
- With Windows XP Professional and Windows Server 2003 regression testing, same problem exists.
- Mac OSX (uses Racoon) – could not reproduce problem
- This does not appear to happen between 2 Linux boxes. We are trying to reproduce the tests where it did.

With the 'Traffic Over SA' tests we used hardware that had Gigabit NICs that used either the Tigon driver or the Intel Driver.

8 Verifying and Troubleshooting IPsec Connections

Due to the complexity of IPsec and IKE protocols, troubleshooting interoperability can be a significant challenge. When conducting the interoperability testing we used a variety of tools to capture and analyze the IPsec traffic on both the Linux and Windows Vista sides. These tools helped us verify the stages of an IPsec connection, from Phase 1 negotiation to showing the ESP and/or AH traffic over the wire.

8.1 Network Captures

Network captures were recorded for all of the tests we performed. For the most part, these captures were done on the Linux host using tcpdump. Tethereal/Wireshark and Netmon were also used to capture and convert traffic dumps. These captures were used for troubleshooting, debugging and verification that the traffic was IPsec protected.

Network capture programs used:

- tcpdump - Linux text based program.
- tethereal - Similar to tcpdump, text based version of Ethereal. Useful for converting tcpdump captures to Netmon format.
- Ethereal/Wireshark – Windows and Linux GUI program, similar to Netmon.
- Netmon Beta 3 (Microsoft Network Monitor) – Windows network capture program.

8.1.1 Using tcpdump

Using tcpdump to verify that an IPsec negotiation was successful and traffic is protected is fairly straight forward. For example; if we wanted to capture the network traffic between hosts 192.168.0.1 and 192.168.0.2, we would run tcpdump with these options:

```
[root@host]# tcpdump -i eth0 -n -vvv -s 1514 'host 192.168.0.1' -w tcpdump.log
```

This tells tcpdump to capture on interface eth0 (-I eth0) no DNS lookup (-n), be very verbose, (-vvv), grab 1514 bytes (-s), only capture packets for host 192.168.0.1 (Expression) and write output to file tcpdump.log (-w). For a full detail of tcpdump options and valid expressions see the manual page tcpdump(8).

Once tcpdump is up and running, open another terminal window or session and ping the remote peer to start the negotiation. Remember you will need to ping twice since the first attempt will error out as the negotiation happens.

```
[root@host]# ping –c 4 192.168.0.2
connect: Resource temporarily unavailable
[root@host]# ping –c 4 192.168.0.2
64 bytes from 192.168.0.2: icmp_seq=0 ttl=127 time=0.548 ms
64 bytes from 192.168.0.2: icmp_seq=1 ttl=127 time=0.385 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=127 time=0.348 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=127 time=0.427 ms
```

Tcpdump logs in a binary format and you will need to use tcpdump's –r option to tell it to read from a file instead of an interface.

```
[root@host]# tcpdump -r tcpdump.log
```

One of the really nice things about saving captures to a log file is that you can still use an expression to filter the output you want to see. For example you only want to see IKE and IPsec related traffic out of a capture.

```
[root@host]# tcpdump  -r tcpdump.log '(udp port 500||ip proto 50)'
```

So to verify that our ping requests and responses were IPsec protected we can look for IP protocol 50 traffic in our capture log. Below is an example of what you would see:

```
[root@host]# tcpdump -nvvv -r tcpdump.log '(ip proto 50)'
18:28:01.374915 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], proto 50,
length: 120) 192.168.0.1 > 192.168.0.2: ESP(spi=0x6c20c667,seq=0x1)
18:28:01.375403 IP (tos 0x0, ttl 127, id 31398, offset 0, flags [none], proto 50,
length: 120) 192.168.0.2 > 192.168.0.1: ESP(spi=0x0116c8d8,seq=0x1)
18:28:02.374531 IP (tos 0x0, ttl  64, id 1, offset 0, flags [DF], proto 50,
length: 120) 192.168.0.1 > 192.168.0.2: ESP(spi=0x6c20c667,seq=0x2)
18:28:02.375065 IP (tos 0x0, ttl 127, id 31400, offset 0, flags [none], proto 50,
length: 120) 192.168.0.2 > 192.168.0.1: ESP(spi=0x0116c8d8,seq=0x2)
18:28:03.374369 IP (tos 0x0, ttl  64, id 2, offset 0, flags [DF], proto 50,
length: 120) 192.168.0.1 > 192.168.0.2: ESP(spi=0x6c20c667,seq=0x3)
18:28:03.374851 IP (tos 0x0, ttl 127, id 31401, offset 0, flags [none], proto 50,
length: 120) 192.168.0.2 > 192.168.0.1: ESP(spi=0x0116c8d8,seq=0x3)
18:28:04.374203 IP (tos 0x0, ttl  64, id 3, offset 0, flags [DF], proto 50,
length: 120) 192.168.0.1 > 192.168.0.2: ESP(spi=0x6c20c667,seq=0x4)
```

Tcpdump is also good for verifying that Phase 1 and Phase 2 negotiate. The output below shows a successful Phase 1 (Main Mode) and Phase 2 (Quick Mode) negotiation.

```
18:28:00.050498 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], proto 17,
length: 108) 192.168.0.1.isakmp > 192.168.0.2.isakmp: [bad udp cksum 3283!] isakmp
1.0 msgid  cookie ->: phase 1 I ident:
    (sa: doi=ipsec situation=identity
        (p: #1 protoid=isakmp transform=1
            (t: #1 id=ike (type=lifetype value=sec)(type=lifeduration
value=7080)(type=enc value=3des)(type=auth value=rsa sig)(type=hash
value=sha1)(type=group desc value=modp1024))))
18:28:00.053596 IP (tos 0x0, ttl 127, id 31393, offset 0, flags [none], proto 17,
length: 236) 192.168.0.2.isakmp > 192.168.0.1.isakmp: [udp sum ok] isakmp 1.0
msgid  cookie ->: phase 1 R ident:
    (sa: doi=ipsec situation=identity
        (p: #1 protoid=isakmp transform=1
            (t: #1 id=ike (type=enc value=3des)(type=hash value=sha1)(type=group
desc value=modp1024)(type=auth value=rsa sig)(type=lifetype
value=sec)(type=lifeduration len=4 value=00007080))))
    (vid: len=20 1e2b516905991c7d7c96fcbfb587e46100000005)
    (vid: len=16 4a131c81070358455c5728f20e95452f)
    (vid: len=16 90cb80913ebb696e086381b5ec427b1f)
    (vid: len=16 4048b7d56ebce88525e7de7f00d6c2d3)
    (vid: len=16 fb1de3cdf341b7ea16b7e5be0855f120)
    (vid: len=16 e3a5966a76379fe707228231e5ce8652)
18:28:00.069620 IP (tos 0x0, ttl  64, id 1, offset 0, flags [DF], proto 17,
length: 208) 192.168.0.1.isakmp > 192.168.0.2.isakmp: [bad udp cksum c5fc!] isakmp
1.0 msgid  cookie ->: phase 1 I ident:
    (ke: key len=128
1d3d67732f9d260df2711107b52bb4628c010936647a5ef082a49b8e841eeee65223efcc342a29d282
2688b6285bc14e62dbf130566ce8d91a807032f6b8865fdfe0c6b09faec7b806d01fd63a1fe30ee81f
ab0bfcfb8b0df1534d14092282acd5a546945993ecf0f666b9bb61556243ee4560c97d15a7487593e1
37c0d58aeb)
    (nonce: n len=16 f070b6ff1c7c079ddf72e5cacfc5af21)
18:28:00.110305 IP (tos 0x0, ttl 127, id 31394, offset 0, flags [none], proto 17,
length: 346) 192.168.0.2.isakmp > 192.168.0.1.isakmp: [udp sum ok] isakmp 1.0
msgid  cookie ->: phase 1 R ident:
    (ke: key len=128
62666a2506b2116f6e6cf15419d76b3b327f9f51809676d569c19d6a996f9e2ee95436676784946c99
954044bc9dd2a44210db6fb7fbdb0db5c4ba5fdb1418826d752f28742a997bdf7becc53b0c65909bee
a17225500a4e8b1845543331ab0ba716fa3a5264ffbafbf4987f63cdcc26937613703d7ebca7b0a85a
e4e44a6cac)
    (nonce: n len=48
23a54e3908da882f97d878c6fc9e5d4b83ff79686124cc981544243b58c99ddd208f94a89110e30826
2a928b1c78821e)
    (cr: len=102 type=x509sign
04306331153013060a0992268993f22c64011916056c6f63616c311d301b060a0992268993f22c6401
19160d4950536563436572745365727631 2b3029060355040313224354414320495053454320436572
7469666963617465205365727665722052 4f4f54)
18:28:00.134618 IP (tos 0x0, ttl  64, id 61853, offset 0, flags [+], proto 17,
length: 1500) 192.168.0.1.isakmp > 192.168.0.2.isakmp: [bad udp cksum d5f!] isakmp
1.0 msgid  cookie ->: phase 1 I ident[E]: [encrypted id] (len mismatch: isakmp
1916/ip 1472)
18:28:00.134627 IP (tos 0x0, ttl  64, id 61853, offset 1480, flags [none], proto
17, length: 464) 192.168.0.1 > 192.168.0.2: udp
18:28:00.163517 IP (tos 0x0, ttl 127, id 31395, offset 0, flags [+], proto 17,
length: 1500) 192.168.0.2.isakmp > 192.168.0.1.isakmp: [bad udp cksum 79ca!]
isakmp 1.0 msgid  cookie ->: phase 1 R ident[E]: [encrypted id] (len mismatch:
isakmp 1820/ip 1472)
18:28:00.163526 IP (tos 0x0, ttl 127, id 31395, offset 1480, flags [none], proto
17, length: 368) 192.168.0.2 > 192.168.0.1: udp
18:28:00.181135 IP (tos 0x0, ttl  64, id 2, offset 0, flags [DF], proto 17,
length: 112) 192.168.0.1.isakmp > 192.168.0.2.isakmp: [bad udp cksum 55ea!] isakmp
1.0 msgid  cookie ->: phase 2/others I inf[E]: [encrypted hash]
```

```
18:28:01.182896 IP (tos 0x0, ttl  64, id 3, offset 0, flags [DF], proto 17,
length: 152) 192.168.0.1.isakmp > 192.168.0.2.isakmp: [bad udp cksum 8677!] isakmp
1.0 msgid  cookie ->: phase 2/others I oakley-quick[E]: [encrypted hash]
18:28:01.185787 IP (tos 0x0, ttl 127, id 31396, offset 0, flags [none], proto 17,
length: 256) 192.168.0.2.isakmp > 192.168.0.1.isakmp: [udp sum ok] isakmp 1.0
msgid  cookie ->: phase 2/others R oakley-quick[EC]: [encrypted hash]
18:28:01.191115 IP (tos 0x0, ttl  64, id 4, offset 0, flags [DF], proto 17,
length: 88) 192.168.0.1.isakmp > 192.168.0.2.isakmp: [bad udp cksum 1560!] isakmp
1.0 msgid  cookie ->: phase 2/others I oakley-quick[EC]: [encrypted hash]
18:28:01.193158 IP (tos 0x0, ttl 127, id 31397, offset 0, flags [none], proto 17,
length: 104) 192.168.0.2.isakmp > 192.168.0.1.isakmp: [udp sum ok] isakmp 1.0
msgid  cookie ->: phase 2/others R oakley-quick[EC]: [encrypted hash]
```

8.1.2 Tethereal / Wireshark

Tethereal is the text based version of Ethereal and was used to convert tcpdump captures to the Netmon format suitable for viewing with Netmon.

```
[root@host]# tethereal -r tcpdump.log -F netmon2 -w netmon.cap
```

8.2 Racoon Log Files

Racoon logs contain valuable information when it comes to debugging and troubleshooting connections. For the testing we set racoon to the maximum logging allowed and saved the output of each test to a file named racoon.log:

```
[root@host]# racoon -v -ddd -f /etc/racoon/racoon.conf -l /etc/racoon/racoon.log
```

A line in racoon.log contains the following information.

- Date - Date in Y-M-D format
- Time - Time in H:M:S format
- Type - INFO, DEBUG, WARNING, or ERROR
- Message - Message data may include packet dump and span multiple lines.

```
2006-07-26 18:27:55: INFO: @(#)ipsec-tools 0.3.3 (http://ipsec-ools.sourceforge.net)
2006-07-26 18:27:55: DEBUG: call pfkey_send_register for AH
2006-07-26 18:27:55: DEBUG: call pfkey_send_register for ESP
2006-07-26 18:27:55: DEBUG: call pfkey_send_register for IPCOMP
```

Some common errors we saw when troubleshooting Phase1 and Phase2:

```
2006-08-02 10:24:34: ERROR: phase1 negotiation failed due to time up.
07051ef0946c928c:c958811c0d71c275
2006-08-02 10:24:47: ERROR: phase2 negotiation failed due to time up waiting for
phase1. ESP 192.168.0.2->192.168.0.1
2006-08-02 10:36:35: ERROR: ignore information because ISAKMP-SA has not been
established yet.
```

If you miss a semi-colon racoon will let you know:

```
2006-08-01 13:17:32: ERROR: /etc/racoon/racoon.conf:15: ";" syntax error
```

It was also helpful to search for certain keywords, other than ERROR, when looking for what went wrong when a negotiation failed. Searching for 'established', 'proposal', 'matched', and 'WARNING' were used quite a lot when we were looking through the logs. It should be noted that there are a couple of misspellings that we came across in the racoon log file (e.g. mismatched is misspelled 'mismathed'). We came across this when we were troubleshooting a proposal error between Windows Vista and Linux.

```
2006-08-02 16:31:54: ERROR: proto_id mismathed: my:2 peer:3
2006-08-02 16:31:54: ERROR: proposal mismathed.
```

8.3 SAD / SPD

Viewing the Security Association Database (SAD)and Security Policy Database or (SPD) is done with the 'setkey' command.

To view the SAD:

```
[root@host]# setkey -D
192.168.0.2 192.168.0.1
        esp mode=transport spi=20226805(0x0134a2f5) reqid=0(0x00000000)
        E: 3des-cbc  1484e628 f8b05eaa 1943cc79 61bb4bbd dcc5f2bb 299dcfcf
        A: hmac-sha1  0346b959 ae6ba87f 2c5f7d30 a97e1ec3 eea280c5
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Aug 18 11:05:25 2006   current: Aug 18 11:10:18 2006
        diff: 293(s)    hard: 3600(s)   soft: 2880(s)
        last: Aug 18 11:05:26 2006      hard: 0(s)      soft: 0(s)
        current: 117788(bytes)  hard: 0(bytes)  soft: 0(bytes)
        allocated: 3414 hard: 0 soft: 0
        sadb_seq=4 pid=7500 refcnt=0
```

To view the SPD:

```
[root@host]# setkey -DP
192.168.0.2[any] 192.168.0.1[any] any
        in ipsec
        esp/transport//require
        created: Aug 18 10:59:46 2006  lastused: Aug 18 11:05:36 2006
        lifetime: 0(s) validtime: 0(s)
        spid=1504 seq=10 pid=7501
        refcnt=1
```

Using the Linux watch command is a nice way to monitor the SAD during a test.

```
[root@host]# watch sektey -D
```

9 Appendix

9.1 Converting Windows Vista PFX files (PKCS#12) For Use with Racoon

For all of the following steps you should be in the /etc/racoon/certs directory.

Extract the client certificates from the .pfx:

```
[root@host]# openssl pkcs12 -in oakleykeylen.pfx -out oakleykeylen.pem -nodes -
clcerts -nokeys
```

Extract the private client key from the .pfx:

```
[root@host]# openssl pkcs12 -in oakleykeylen.pfx -out oakleykeylenca.key -nodes -
nocerts
```

Extract the CA's certificate:

```
[root@host]# openssl pkcs12 -in oakleykeylen.pfx -out oakleykeylenca.pem -nodes -
cacerts -nokeys
```

#

Make the root certificate trusted:

```
[root@host]# ln -s oakleykeylenca.pem `openssl x509 -noout -hash -in
oakleykeylenca.pem`.0
```