# Contents

# 1 Introduction

This is an implementation of Secure Neighbor Discovery (SEND) for IPV6.
The primary goal of this project is to reveal flaws in SEND by creating an implementation of the protocol. Therefore, in order to make the implementation as compatible to RFC3971 specifications as possible, SEND is implemented within Linux kernel IPv6 module, where direct access to neighbor caches and routing tables are available.
Most of the encryption and decryption related calculating works, such as CGA generation/verification, signature generation/verification, and X.509 certificate processing are required by SEND, but the Linux kernel cannot provide enough support. Therefore, we implement a daemon process in user-space, which makes use of Openssl package, to perform these calculating works.
This is a research prototype still under development, so there are no doubt bugs—bugs that sometimes could even cause kernel crashes. Lots of works are still needed to be done to provide both robustness and stability. Do not expect any commercial-grade reliability and security.

# 2 Feature

SEND supports the following RFCs:

- RFC3971: Secure Neighbor Discovery (SEND)
- RFC3972: Cryptographically Generated Addresses (CGAs)
- RFC3779: X.509 Extensions for IP Address and AS Identifiers

Along with RSA algorithm (which specified by RFC3971), ECC algorithm is implemented as an alternative signature algorithm. A simple CRL verification mechanism is also provided.
Please refer to related RFCs for more details about SEND protocol and related algorithms.

# 3 Supported Platforms

SEND is developed on Ubuntu 8.04 with 2.6.24 kernel. It is recommended to install SEND on an Ubuntu 8.04 machine with 2.6.24 kernel.
To run SEND your kernel must have the following enabled:

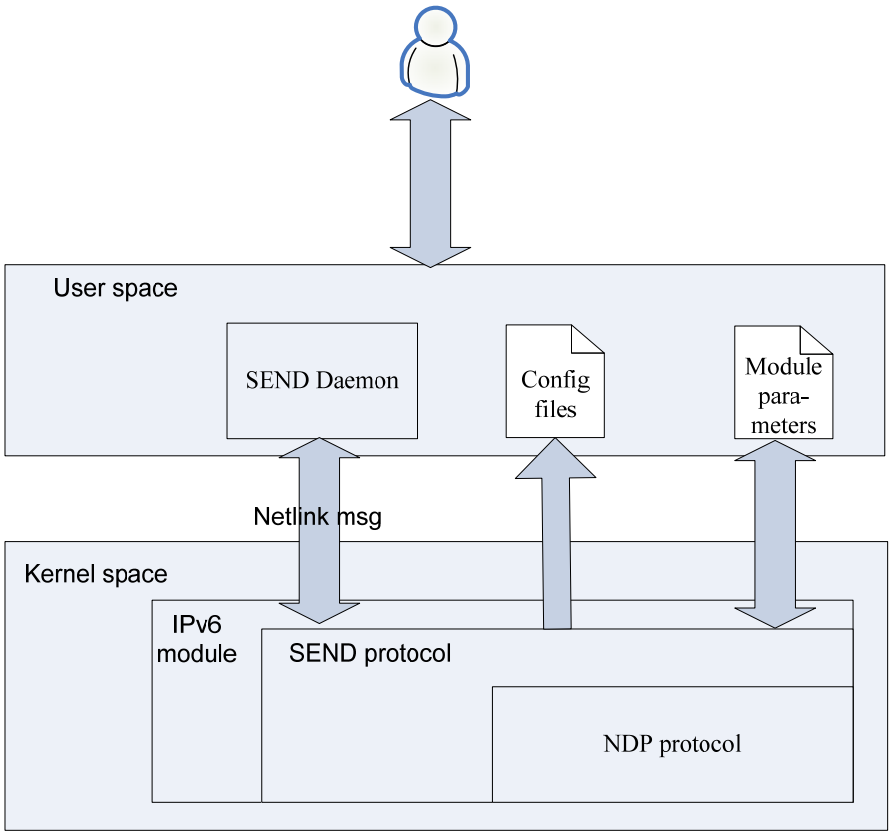
- CONFIG_NETFILTER
- CONFIG_IPV6
- CONFIG_IP6_IPTABLES
- CONFIG_IP6_NF_FILTER

Also the following libraries and packages are required:

- libpthread
- libnl-dev
- GNU make
- gcc
- pkg-config
- Openssl 0.9.8j (or higher version)

# 4 Installation

SEND is composed by following parts: a SEND patched ipv6 module in Linux kernel, SEND daemon process in user space, and some configuration files.




Thus the installation of SEND contains following two steps:

- Compile a SEND patched IPV6 kernel module
- Compile and setup the SEND daemon in user-space

## 4.1 Kernel Module Installation

The kernel module installation includes following steps:

- Download and compiling a custom kernel
- Patch the kernel source code
- Compile the SEND patched IPV6 kernel module

### 4.1.1 Compiling a Custom Kernel

You will need to download the Linux kernel source code and compile it first before installing SEND kernel module. There is lots of information in the internet about how to compile a Linux kernel, and you could always refer to forums of distribution (such as, Ubuntu forum) for instructions. Here is a URL where details of compiling kernels on Ubuntu are provided:
http://www.howtogeek.com/howto/ubuntu/how-to-customize-your-ubuntu-kernel/

After compiling the kernel, reboot your machine and enter into the newly compiled kernel.

### 4.1.2 Patching Kernel Source Code

You need to copy the *include/* folder and *net/* folder in *sendcgaknl/* to your linux source tree (where you compiled your kernel), and copy the *sndd/* folder in *sendcgaknl/* to */etc/* folder.
**Note that** you will need root privilege to accomplish the above.

### 4.1.3 Compiling IPv6 Module

You can compile the SEND kernel module by executing the commands below:

*$ cd [ your Linux source tree ]/net/ipv6*

*$ make*

As a result a kernel dynamic link library “ipv6.ko” should be created under the folder.

## 4.2 SEND Daemon Installation

### 4.2.1 Installation of Openssl

Openssl or libcrypto.so might have already been installed in your system (for most Linux distribution they are installed by default), but a re-installation of the Openssl package is still required – this is because SEND daemon needs the rfc3779 support of Openssl package to handle the IP address extension of X509 certificates, but that support is disabled by default.
Here is a quick start of the installation of Openssl: download Openssl package (version 0.9.8j or greater) from http://www.openssl.org/, decompress it, enter the decompressed folder, and execute:

*$ ./config enable-rfc3779 shared*

*$ make depend*

*$ make*

*$ make test*

*$ make install*

**Note that** you might need root privilege to execute “make install” command. The option “*enable-rfc3779*”of “*./config*” command means to enable rfc3779 support, and “*shared*” means to create shared library which is more convenient to use. For more details of the installation, please refer to the Openssl installation document.

### 4.2.2 Installation of SEND Daemon

Assuming that the Openssl package is installed at */usr/local/ssl/* (it is the default installation directory), you must execute the following command to assure that pkg-config can find the newly installed Openssl package during compilation:

*$ export PKG_CONFIG_PATH=/usr/local/ssl/lib/pkgconfig/:$PKG_CONFIG_PATH*

Without executing this “export” command, the “./configure” below will might failed because pkg-config cannot locate the Openssl package. **Note that** the “export” command above would only take effect within the shell it’s been executed; you shall add the command into your shell’s

startup script if you want it to take effect in every shell.

The SEND daemon can then be installed by executing following commands:

*$ ./configure*

*$ make*

*$ make install*

**Note that** you might need root privilege to execute "*make install*" command.

### 4.2.3 Installation of AddIpExt Tool

The addIpExt tool which is used to add IP address extensions to specified X509 certificates will also be installed during the installation of SEND daemon. For details about the usage of addIpExt tool, please refer to Chapter 7 of this document.

## 5 Running SEND

In current release SEND is in an ipv6 module that can only be inserted during runtime. Therefore, you should blacklist the ipv6 module in kernel before insertion. In Ubuntu 8.04, this could be achieved by following steps:

1. *open the file /etc/modprobe.d/aliases*
2. *modify the line "alias net-pf-10 ipv6" to "alias net-pf-10 off"*
3. *add a line "blacklist ipv6" to the bottom of the file /etc/modprobe.d/blacklist*
4. *reboot to apply changes*

Now you can start SEND by following commands:

1. *cd [ your Linux source tree ]/net/ipv6*
2. *insmod ipv6.ko*
3. *sendd &*

**Note that** the SEND daemon must be started after the kernel IPV6 module insertion. **Also note that** in the pre-installed Openssl package (or libcrypto.so) of most Linux distributions, the rfc3779 support is disabled by default. Therefore, you need make sure that the SEND daemon will link to the rfc3779-enabled libcrypto.so (which is installed by following the instructions in section.3.2.2) during runtime. This could be achieved by executing the following command (assuming Openssl package is installed at */usr/local/ssl/*):

*$export LD_LIBRARY_PATH=/usr/loca/ssl/lib/:$LD_LIBRARY_PATH*

As pointed out in 4.2.2, this *export* command would only take effect within the shell it's been executed, you shall add the command into your shell's startup script if you want it to take effect in every shell.

**Note that** the logging messages of both the SEND kernel module and SEND daemon will be logged into the file /var/log/syslog.

## 6 Configurations

There are two parts of configurations of SEND – module parameters and configuration files. Module parameters control the protocol behavior during runtime, and configuration files provide

essential information for SEND on module initiation.
Details about module parameters and configuration files are given in this part, and examples of configuration files are provided in *sendcgacknl/sndd/* folder in the tarball.

## 6.1 Module Parameters

Some parameters in SEND kernel module are declared as module parameters by using Sysfs. These parameters can be modified during runtime to dynamically control protocol behaviors of SEND.
In Ubuntu system these parameters can be found under the directory /sys/module/ipv6/parameters after the module has been inserted. They can be modified by using the "echo" command as below:

*echo –n "1" > /sys/module/ipv6/parameters/snd_use_snd*

By executing this command, the value of parameter "snd_use_snd" is set to 1.
These module parameters are:

| Name | Description | Default Value | Reference |
|---|---|---|---|
| Snd_use_snd | Disable the use of SEND completely if set to 0 | 1 | RFC3971, 8 |
| Snd_ignore_unsec | Cause SEND to ignore all the unsecured ND messages if set to 1 | 0 | RFC3971, 8 |
| Snd_ignore_unsec_adv | Cause SEND to ignore all the unsecured advertisements if set to 1 | 0 | RFC3971, 8 |
| Snd_is_router | Inform SEND that local machine is functioning as a router if set to 1 | 0 | |
| Snd_check_addr_ext | Cause SEND to do rigorous address check according to address extension field in certificate if set to 1 | 1 | RFC3971, 6.3.1 |

## 6.2 Configuration Files

SEND also has some configuration files to provide information to kernel module. These files are loaded into kernel module during module insertion. Any changes to configuration files during runtime will not take effect until the system is rebooted. Comments in these files are started with the character '#'.
These configuration files include:

### */etc/sndd/cfg/snd_cfg*

Inform kernel module the SEND protocol value. A valid line in snd_cfg takes the form [ parameter ]=[ value ]. The accepted parameters are:

***cps_retry***: CPS_RETRY in RFC3971, 10.1. Only integers are accepted.

***cps_retry_fragments***: CPS_RETRY_FRAGMENTS in RFC3971, 10.1. Only integers are accepted.

***cps_retry_max***: CPS_RETRY_MAX in RFC3971, 10.1. Only integers are accepted.

***max_cpa_rate*:** MAX_CPA_RATE in RFC3971, 10.1. Only integers are accepted.

***timestamp_delta*:** TIMESTAMP_DELTA in RFC3971, 10.2. Only integers are accepted.

***timestamp_fuzz*:** TIMESTAMP_FUZZ in RFC3971, 10.2. Only integers are accepted.

***timestamp_drift***: TIMESTAMP_DRIFT in RFC3971, 10.2. Only integers are accepted.

***nonce_lft***: specifies the lifetime (in seconds) of a sent-out nonce in cache. The sent-out nonce is deleted from cache if this lifetime is exceeded and no responds with that nonce received within the period. If not specified, the default value of it is 5 seconds. Only integers are accepted.

***max_cpa_buff*:** numbers of CPAs that each ADD process can cache. If a received CPA could not be verified immediately, it is cached in a queue of the ADD process. Only integers are accepted.

***is_router***: provides a default value for the module parameter *snd_is_router*. The available values include 0 and 1. **Note that** if not specified, the default value for *snd_is_router* is 0, as mentioned above.

### */etc/sndd/snd_ta*

Configure the paths of the trust anchors and certificate paths on local machine. For host nodes, providing paths of trust anchors are sufficient:

```
{
trust_anchor=[ path1 ]
}
{
trust_anchor=[ path2 ]
}
…..
```

**Note that** the path provided should be a complete path starting from root (/), and the double quotation marks are necessary. For example, a valid configuration for trust anchor path could be:

*trust_anchor=“/etc/sndd/ta/ta1.pem”*

For router nodes, both trust anchors and certificate paths should be provided, and matching trust anchor and certificate path should be provided together:

```
{
trust_anchor=[ path1 ]
cert=[ path2 ]
cert=[ path3 ]
cert=[ path4 ]
}
{
trust_anchor=[ path5 ]
cert=[ path6 ]
}
…..
```

**Note that** certificate path **MUST** be provided in correct order. For instance, in the above certificate path of 3 components: the trust anchor in *[ path1 ]* is a self-signed certificate of CA1; certificate in *[ path2 ]* is signed by CA1 for R1; certificate in *[ path3 ]* in signed by R1 for R2; and

the last certificate in *[ path4 ]* is signed by R2 for local machine.
**Also note that** the provided trust anchors should be of PEM form, and certificates should be DER encoded.

### */etc/sndd/iface/[ iface_name ].conf*

This configuration file provides the sec value of parameter and the private/public key pair for a network interface of the local machine. If local machine has a network interface eth0, then a configuration file named eth0.conf should be created and set as following:

*sec=1*
*{*
*public_key=[ path1 ]*
*pub_key_code=DER*
*pub_key_type=ECC*
*private_key=[ path2 ]*
*pri_key_code=DER*
*pri_key_type=ECC*
*}*
…

*Pub_key_code/pri_key_code* informs the format (PEM or DER) of the public/private key. *Pub_key_type/pri_key_type* informs the type (RSA or ECC) of the key pair – RSA is the type that specified in RFC3971, and ECC (also represented as ECDSA in Openssl) stands for Elliptic Curve Cryptography. **Note that** public key and private key must be provided in pair, and key_type and key_code for both the public and private key must be provided.

### */etc/sndd/iface/[ iface_name ].cga*

This is a binary file created/modified by SEND kernel module. The CGA addresses and their corresponding CGA parameters are stored in this file. When rebooted, from this file kernel can retrieve previous generated CGA addresses and apply them to the interface. If you want to regenerate the CGA address for the interface, you can simply delete this file.
**Note that** once the public/private key pair provided in *[ iface_name ].conf* is changed, this file **MUST** be deleted to regenerate CGA address. Otherwise, the SEND message sent out by this interface will never pass the CGA address check.

## 6.3 Router Configuration

As specified above, the module parameter “snd_is_router” must be set to 1 for local machine to function as a router. Moreover, a routing suite, such as Quagga, is also required. Since SEND kernel module itself will not generate RA (Router Advertisement) message, Quagga or other routing suites are needed to send out RA.
**Note that** although SEND kernel module won’t generate RA message, it adds SEND options to RA message created by Quagga and send them out. It captures the RA messages generated by Quagga using a ‘local-out hook’ provided by IPV6_NET_FILTER, adds SEND options to the messages, and then send them out.
Lots of protocol behaviors (such as, the interval between RA sending) can be specified in Quagga.

You can refer to this site for details of the configuration:
http://www.quagga.net/docs.php

## 6.4 CRL Checking Configuration

SEND daemon supports certificate path verification against CRL. However, in current SEND protocol there are no provided mechanisms for acquiring CRL files. Thus you must explicitly inform SEND daemon the location of the CRL files. This could be accomplished by providing a CRL configuration file – a configuration file composed of the absolute paths of CRL files.
Here is an example of the configuration file: assuming you want to verify certificate path against two CRL files, *crl1.pem* and *crl2.pem* which are in the directory "*/home/crl/*", so the content of the CRL configuration file should be:

```
/home/crl/crl1.pem
/home/crl/crl2.pem
```

When you run SEND daemon, provide a "-c" option and make the parameter the path of the CRL configuration file as the following:

*$sendd –c [crl_conf_file_path] &*

You can modify the CRL configuration file to update the CRL files during SEND daemon runtime, and SEND daemon will scan the CRL configuration file periodically to gain the latest specified CRL files.

# 7 Usage of AddIpExt Tool

The addIpExt tool is used to add IP address extensions to specified X509 certificates. To accomplish this task, an IP address extension configuration file is needed.
Here is a simple example of an IP address extension configuration file.

```
# a comment line should start with '#'

#IPv6 field
[ IPv6 ]
inherit = 0
prefix = my_prefix
range = my_range


[ my_prefix ]
prefix0 = 2001:0:2::/64
prefix1 = 2001:0:3::/64

[ my_range ]
min0 = 2001:0:5::
```

```
max0 = 2001:0:5:0:ffff:ffff:ffff:ffff

#files field
[ files ]
certfile   = /…/cert.pem
cacert     = /…/cacert.pem
capriv     = /…/caprivkey.pem
outfile    = /…/cert_added.pem
```

The detail of the configuration file is given below.
The basic unit of a configuration file is called “filed”, in which there may be several variables. A field’s profile should be like this:

*[ field name ]*
*variable = var_filed_name(or value)*

The field’s name is in the brackets, and there may be several lines below the field name. For each line, there is an equation. On the left of the equation it is the variable’s name and on the right it is the variable’s value or the name of anther filed which represents the variable.

There are two main fields in a configuration file – the ‘IPv6’ field and the ‘files’ field. ‘IPv6’ field specifies the IP address extension which will be added into the specified X509 certificate, while ‘files’ field specifies the locations of the certificate and the key files.

IPv6 field has three variables: ‘inherit’, ‘prefix’ and ‘range’.
Variable ‘inherit’ specifies whether the IP addresses of the IP address extension should inherit from the issuer's certificate. If the value of the variable ‘inherit’ is set to 1, there should be no ‘prefix’ or ‘range’ variables, and the IP addresses of IP address extension of the certificate should inherit from the issuer's certificate.

If the value of the variable ‘inherit’ is set to 0, you must provide variable 'prefix' or 'range' or both. For each variable, there should be a field which represents the variable. For the prefix’ filed, each variable of the field should have a value which is a subnet prefix formatted as specified by RFC4291. For range’s field, each of the variables of this field has an IPv6 address as its value, and each of the two consecutive variable pair makes one range. Please refer to the example above.

Files field has four variables: certfile, cacert, capriv and outfile, which specify the location of the original certificate which the IP address extension will be added into, the location of the certificate of the original certificate’s issuer, the location of private key of the issuer, and the location of the resulted certificate respectively. **Note that** the locations should be the absolute path of the files.

After the IP address extension configuration file is ready, we can add IP address extension into a specified X509 certificate. The process is rather easy, just execute:

*$addIpExt [IP_add_exten_conf_file]*

**Note that** maybe you would need an ‘export’ command to make sure the addIpExt program will

link the proper libcrypto.so at runtime, please refer to Chapter 5 of this document.
And there can be more than one configure file as parameters in order to add IP address extensions for more than one X509 certificates.

You can refer to the document README in the ‘cert_path/’ directory for more detail about certificate path.