Easy-SEND: A Didactic Implementation of the Secure Neighbor Discovery Protocol for IPv6

Say Chiu and Eric Gamess

***Abstract*—IPv6 adds many improvements to IPv4 in areas such as address space, built-in security, quality of service, routing and network auto-configuration. IPv6 nodes use the Neighbor Discovery (ND) protocol to discover other nodes on the link, to determine their link-layer addresses, to find routers, to detect duplicate address, and to maintain reachability information about the paths to active neighbors. ND is vulnerable to various attacks when it is not secured. The original specifications of ND called for the use of IPsec as a security mechanism to protect ND messages. However, its use is impractical due to the very large number of manually configured security associations needed for protecting ND. For this reason, the Secure Neighbor Discovery Protocol (SEND) was proposed. In this paper, we present Easy-SEND, an open source implementation of SEND that can be used in production environment or as a didactic application for the teaching and learning of the SEND protocol. Easy-SEND is easy to install and use, and it has an event logger that can help network administrators to troubleshoot problems or students in their studies. It also includes a tool to generate and verify Cryptographically Generated Addresses (CGA) that are used with SEND.**

***Index Terms*— Cryptography, CGA, IPv6, Secure Neighbor Discovery Protocol, Security.**

I. Introduction

IPv6 [6][7][9] (Internet Protocol version 6) is a solution to the problem of the shortage of public IPv4 addresses that faces Internet. IPv6 adds many improvements to IPv4 in areas such as quality of service, routing and network auto-configuration. Even if IPv6 has been around for more than ten years now, there is a lack of IPv6 network specialists in Venezuela and around the world. Therefore, the training of IPv6 specialists has become an important issue. In the undergraduate program of Computer Science at *Central University of Venezuela* (in Spanish: Universidad Central de Venezuela), some courses have been upgraded or added to the curriculum to face the problem. For example, *Advanced Network Protocols* (in Spanish: Protocolos Avanzados en Redes) is a new course that was introduced to the curriculum of the undergraduate program of Computer Science in 2005. Its objectives include the understanding of IPv6 standards, such as the ND [13] (Neighbor Discovery) protocol and the SEND [1] (Secure Neighbor Discovery) protocol.

Manuscript received July 7, 2009.

S. Chiu is with the School of Computer Science, Universidad Central de Venezuela, Facultad de Ciencias, Los Chaguaramos, Caracas, Venezuela (e-mail: saychiu@gmail.com).

E. Gamess is with the Laboratory of Communications and Networks, Universidad Central de Venezuela, Facultad de Ciencias, Caracas, Venezuela (phone: +58-212-6051061; e-mail: egamess@kuaimare.ciens.ucv.ve).

Since ND [13] is supported by all the actual modern operating systems, a variety of practices are done in the course (Advanced Network Protocols) to strengthen student's knowledge about this important protocol. However, we have been facing the lack of support for SEND [1] (as stated is Section V) from manufacturers and it has been almost impossible for us to do laboratories to clarify the complex procedures involved in SEND. Therefore, we decided to develop a new application (*Easy-SEND*) from scratch that implements the SEND protocol, with good support for the teaching and learning process. Its main goal is to be used as a didactic application in advanced courses related to networks at *Central University of Venezuela*. Additionally, *Easy-SEND* can be used in production networks where security is important as a replacement of the ND protocol.

The rest of this paper is organized as follows: An overview of ND is presented in Section II. Vulnerability issues for ND are discussed in Section III. SEND is presented in Section IV. Related works are viewed in Section V. *Easy-SEND* is introduced and justified in Section VI. Conclusions and future work are discussed in Section VII.

II. Neighbor Discovery Protocol Overview

The ND [13] (Neighbor Discovery) protocol solves a set of problems related to the interaction between nodes attached to the same link. It defines mechanisms to solution each of the following problems: router discovery, prefix discovery, parameter discovery, address auto-configuration, address resolution, next-hop determination, Neighbor Unreachability Detection (NUD), Duplicate Address Detection (DAD), and redirect.

The ND protocol has five messages or PDUs (Protocol Data Units) provided by ICMPv6 [5] (Internet Control Message Protocol version 6), an updated version of ICMPv4 [8] (Internet Control Message Protocol version 4). These messages are: RS (Router Solicitation), RA (Router Advertisement), NS (Neighbor Solicitation), NA (Neighbor Advertisement), and Redirect. RS are sent by IPv6 hosts to discover neighboring routers on an attached link. RA are sent by IPv6 routers periodically (unsolicited multicast router advertisements) or in response to a RS message (solicited router advertisements). NS are sent by IPv6 nodes to resolve a neighbor's IPv6 address to its link-layer address (MAC address) or to verify if an IPv6 node is still reachable (NUD). NA are sent by IPv6 nodes in response to a NS message or to propagate a link-layer address change. Redirect messages are sent by IPv6 routers to inform hosts of a better first-hop for a destination.

ND messages consist of an ND message header, composed of an ICMPv6 header, some ND message-specific data, and zero or more ND options. Fig. 1 shows the format of an ND message.

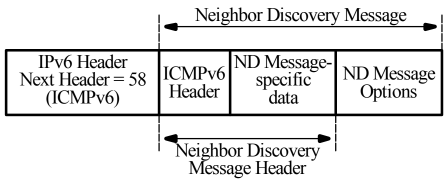

Neighbor Discovery Message
IPv6 Header Next Header = 58 (ICMPv6)
ICMPv6 Header
ND Message-specific data
ND Message Options
Neighbor Discovery Message Header

Fig. 1: Format of an ND message

ND message options provide additional information, such as link-layer addresses, on-link network prefixes, on-link MTU information, redirection data, mobility information, and specific routes. These options are formatted in type-length-value (TLV) format. Table I shows the main options of the ND protocol with a brief description.

Table I: Options for Neighbor Discovery

Type	Option Name	Description
1	Source Link-Layer Address	Included in NS, RS and RA messages to indicate the link-layer address of the sender of the packet.
2	Target Link-Layer Address	It indicates the link-layer address of the target. It is used in NA and Redirect messages.
3	Prefix Information	Appear in RA messages to provide hosts with on-link prefixes and prefixes for address auto-configuration.
4	Redirected Header	It is used in Redirect messages and contains all or part of the packet that is being redirected.
5	Maximum Transmission Unit (MTU)	This option is used in RA messages to ensure that all nodes on a link use the same MTU value in those cases where the link MTU is not well known.

III. Vulnerabilities of the Neighbor Discovery Protocol

ND can suffer different attacks that can be classified in three types as follow:

- Impersonation/Spoofing: class of attacks in which a malicious node successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage. This type of attacks is easily made since there is no control over MAC addresses and they can be change with little effort.
- DoS (Denial of Service): type of attacks in which a malicious node prevents communication between the node under attack and all other nodes.
- Redirection: class of attacks in which a malicious node redirects packets away from the legitimate receiver to another node on the link.

According to [14], the ND protocol suffers frequent attacks that include, but are not limited to: Neighbor Solicitation Spoofing, Neighbor Advertisement Spoofing, Neighbor Unreachability Detection Failure, Duplicate Address Detection DoS Attack, Malicious Last Hop Router, Spoofed Redirect Message, Bogus On-Link Prefix, Bogus Address Configuration Prefix, Parameter Spoofing, Replay Attacks, and Neighbor Discovery DoS Attack.

IV. Secure Neighbor Discovery Protocol

As stated in Section III, ND is insecure and susceptible to malicious interference. To counter the threats of the ND protocol, IPsec [11][12] (Internet Protocol Security) can be used. One of the main objectives of ND is the auto-configuration of hosts with no human intervention. With IPsec, the key exchange can be done automatically when hosts already have a valid IPv6 address. Without an initial IPv6 address, the usage of IPsec can be quite tedious and impracticable. For these reasons, the IETF (Internet Engineering Force) developed the SEND [1] (Secure Neighbor Discovery) protocol.

SEND is a security extension of the ND protocol. It introduces new messages (CPS and CPA) and options (CGA, Timestamp, Nonce, RSA Signature, Trust Anchor and Certificate) to the ND protocol, as well as defense mechanisms against attacks on integrity and identity. The main components used in the SEND protocol are:

- Certification paths: anchored on trusted parties certify the authority of routers. A host must be configured with a trust anchor to which the router has a certification path before the host can adopt the router as its default router. CPS (Certification Path Solicitation) and CPA (Certification Path Advertisement) messages are used to discover a certification path to the trust anchor.
- Cryptographically Generated Addresses (CGA): are used to make sure that the sender of a Neighbor Discovery message is the owner of the claimed address. A public-private key pair is generated by all nodes before they can claim an address. A new ND option, the CGA option, is used to carry the public key and associated parameters.
- RSA Signature option: is used to protect all messages relating to Neighbor and Router Discovery.
- Timestamp option: Provides replay protection against unsolicited advertisements and redirects.
- Nonce option: Ensures that an advertisement is a fresh response to a solicitation sent earlier by the node.

Table II: ICMPv6 message for SEND

ICMPv6 Type	SEND Message	Description
148	Certification Path Solicitation (CPS)	Sent by hosts during the ADD process to request a certification path between a router and one of the host's trust anchors.
149	Certification Path Advertisement (CPA)	The CPA message is sent in reply to the CPS message and contains the router certificate.

Similarly to the ND protocol, SEND uses ICMPv6 [5] messages. Two new messages were added for the ADD

(Authorization Delegation Discovery) process. Table II shows the value of the Type field from the ICMPv6 message and a brief description.

SEND defines two mechanisms (CGA and ADD) presented in the following sections for securing the ND protocol.

A. Cryptographically Generated Address

CGAs [3] (Cryptographically Generated Addresses) are IPv6 addresses generated with a hash function, from a public key and auxiliary parameters. This provides a method for securely associating a cryptographic public key with an IPv6 address in the SEND protocol. The node generating a CGA address must first obtain a RSA (Rivest, Shamir, and Adelman) key pair (SEND uses an RSA public/private key pair). The node then computes the interface identifier part (which is the rightmost 64 bits) and appends the result to the prefix to form the CGA address (see Fig. 2).

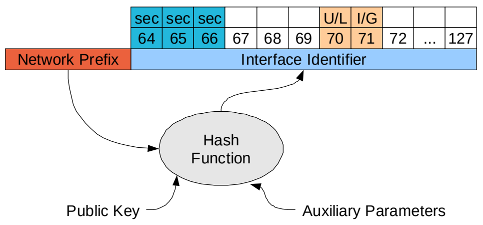


Fig. 2: Generation process of a CGA

B. Authorization Delegation Discovery

The ADD (Authorization Delegation Discovery) is used to certify the authority of routers via a trust anchor. A "trust anchor" is an authoritative source that can be trusted by hosts, and which is used to certify the authority of routers: "certification paths" can be established from the trust anchors to enable a certified router to perform certification of another router; before any host can adopt a router as its default router, the host must be configured with a trust anchor that can certify the router via a certification path. Hence, the certificate format mandated by the SEND protocol enables the hosts to learn a certification path of a router, and to validate the authenticity of a router.

V. Related Works

At the present time, manufacturers of network devices and developers of operating systems do not include functionalities of the SEND protocol in their products. The unique implementation of SEND had been developed by members of the working group that did its specification [1], especially James Kempf who works at DoCoMo USA Labs. DoCoMo's Open Source SEND Project is an implementation of SEND developed by DoCoMo USA Labs that works in the operating system user-space. It can be installed in Linux (kernel 2.6) or FreeBSD 5.4.

The SEND implementation from DoCoMo USA Labs has a limited debugging mode where users can see information related to the internal states and the operations executed by the application. It also includes two tools (*cgatool* and *ipexttool*) to generate keys, CGA addresses and the extension for IPv6 addresses required in X.509 certificates [10]. The application allows two configuration modes: basic and advanced modes. In the basic mode, users can create a SEND environment for hosts only, without on-link router discovery. In the advanced mode, hosts can discover on-link routers by using certification path and their correspondent trust anchor.

The SEND application from DoCoMo USA Labs has some drawbacks. It is not able to manage address collisions during the DAD (Duplicate Address Detection) process, due to the difficulties carried by its integration to the kernel. Additionally, users must have good knowledge in the specification of firewall rules (*ip6tables*), since they are essential for the configuration of the application. Also, DoCoMo USA Labs has announced a few weeks ago that it will no longer maintain the SEND project[1]. Hence, the source code is no longer available to download in the web site and support has been canceled.

JSend[2] is announced in SourceForge as a free and open source implementation of the SEND protocol in Java. Its main objective is to be used for learning and testing purpose. For now, this project has not released any files.

VI. A Didactic Implementation of the SEND Protocol

Easy-SEND is a free and open source application developed in Java that implements the SEND [1] (Secure Neighbor Discovery) protocol. It also includes a tool (*CGAGen*) to generate and verify CGAs [3] (Cryptographically Generated Addresses).

We developed the application in the Linux user-space that does not require modification at the kernel level. *Easy-SEND* works as a firewall between the network interface card and the IPv6 stack, by using the *ip6tables* filtering rules. The actual version is limited to the creation of a secure environment for IPv6 hosts; that is, hosts are not able to participate in the Router Discovery process.

The architecture and the functionality of *Easy-SEND* are summarized in Fig. 3. *libipq*[3] provides a mechanism for passing packets out of the stack for queuing to user-space, then receiving these packets back into the kernel with a verdict specifying what to do with the packets (such as acceptation or rejection). These packets may also be modified in the user-space prior to reinjection back into the kernel.

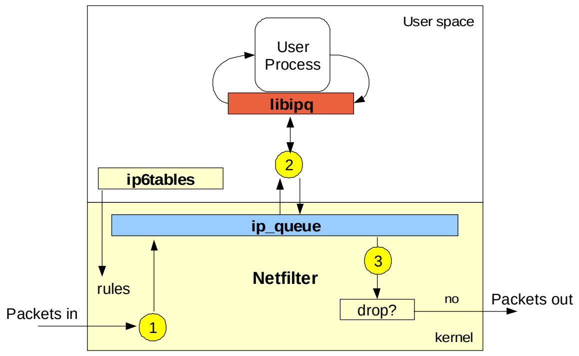


Fig. 3: Easy-SEND architecture

[1] http://www.docomolabs-usa.com/lab_opensource.html
[2] http://sourceforge.net/projects/jsend
[3] http://www.netfilter.org/projects

Initially, ND messages are selected based on *ip6tables* rules (see Fig. 4) and sent to *ip_queue*, a queue managed by *libipq*. From the first to the fifth line of Fig. 4, *ip6tables* rules capture RS (type 133), RA (type 134), NS (type 135), NA (type 136), and Redirect (type 137) messages, respectively. Library *libipq* allows the manipulation of ND packets (aggregation and verification of options) based on the SEND specifications before the determination of the verdict (acceptation or rejection of the packets). Messages marked for rejection will be dropped. Message marked for acceptation will be further processed.

```
ip6tables -A INPUT -p icmpv6 -j QUEUE --icmpv6-type "133"
ip6tables -A INPUT -p icmpv6 -j QUEUE --icmpv6-type "134"
ip6tables -A INPUT -p icmpv6 -j QUEUE --icmpv6-type "135"
ip6tables -A INPUT -p icmpv6 -j QUEUE --icmpv6-type "136"
ip6tables -A INPUT -p icmpv6 -j QUEUE --icmpv6-type "137"
```

Fig. 4: Capture rules for ND messages with ip6tables

Since library *libipq* was written in C, we used a wrapper, called *Virtual Services IPQ*[4] or *VServ IPQ* for short, which allows Java users to call the *libipq* native functions from Java applications.

To develop the application, we used the waterfall methodology [4][16]. It is a sequential methodology, in which development is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Requirement, Analysis, Design, Coding, and Testing. The application's general structure consists of a set of classes that interacts with each others. Fig. 7 shows the principal classes of *Easy-SEND*. Class *SENDOPT* is an abstract class where get and set methods are defined to access and modify SEND options. Class *CGA* allows the generation and manipulation of CGA addresses. Methods of the *RSASignature* class permit the creation and verification of digital signatures. *TimeStamp* and *Nonce* are classes with methods to produce and handle timestamps for the protection against replay attacks. Class *Monitor* is for the capture and manipulation of packets that arrive to the *ip6tables* queue. *NDMessage* is a class for the aggregation (outgoing packets) and removal (incoming packets) of SEND options in messages. Its methods automatically update the checksum field. Class *KeyManagement* allows users to generate, load and save RSA keys.

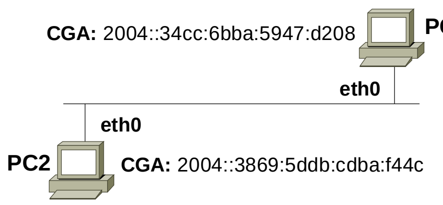

CGA: 2004::34cc:6bba:5947:d208
PC1
eth0
eth0
PC2
CGA: 2004::3869:5ddb:cdba:f44c

Fig. 5: Test-bed topology

To validate our implementation of the SEND protocol, we make several test experiments. For reasons of space, in this paper we will only briefly describe two of our experiments. In the first experiment, we setup a test-bed (see Fig. 5) where we connected two PCs in an Ethernet LAN. Each PC had 512 MB of RAM, and we installed Ubuntu 8.04 LTS (a community developed, Linux-based operating system), Java 2SE 1.6.0_07 and *Easy-SEND*.

Fig. 8 shows the ND messages sent by PC1 and PC2 during the execution of a ping command from PC1 to PC2. We can observe that *Easy-SEND* modifies the original messages (NS and NA) to add SEND options (CGA, Timestamp, Nonce, and RSA Signature). We used Wireshark [15] (a popular network analyzer formerly known as Ethereal) to analyze the SEND protocol.

In the second experiment, we made interoperability tests between *Easy-SEND* and the SEND implementation of DoCoMo USA labs. We made a test-bed similar to the one shown in Fig. 5, but we installed the SEND implementation of DoCoMo USA Labs in PC1 and *Easy-SEND* in PC2. We made different communication tests (FTP, SSH, PING, TFTP, etc) between the two PCs and had a perfect interoperability.

```
[06:53:07,577] INFO Monitor - NetfilterPacket Hook Local Input
[06:53:07,578] INFO  Monitor  - Incoming Interface: eth0
[06:53:07,579] INFO  Monitor  - Verifying SEND options
[06:53:07,579] INFO  Monitor  - ICMPv6 Type: <- Neighbor
Solicitation
[06:53:07,579] INFO  CGA  - Verifying CGA
[06:53:07,580] INFO  CGA  - Verifying address:
fe80:0:0:0:2c8e:4f5b:84b:80c1

[06:53:07,580] DEBUG CGA  - CGA Parameters:

  1b be bd c4 5f 26 a4 c0 66 6b eb 46 73 44 fe 32
  fe 80 00 00 00 00 00 00 00 30 81 9f 30 0d 06 09
  2a 86 48 86 f7 0d 01 01 01 05 00 03 81 8d 00 30
  81 89 02 81 81 00 b0 89 3b 51 88 d3 f2 3c 9b a0
  98 4f 99 01 b4 5f de 81 89 b3 d6 ba 96 ff a7 05
  b3 49 4c 28 ce 60 2e ba e7 f6 76 f4 c2 87 f9 51
  36 be 08 12 ab c2 e5 00 50 8a b5 be fb 04 79 99
  79 b4 ee 88 5b 45 b7 f1 88 a9 e2 e4 52 3a 75 9b
  ea 94 87 38 56 49 8d 63 66 b6 78 1e d0 41 4a f5
  25 e9 cf 69 50 9b f2 b9 68 91 a9 8c e6 8a 6f 61
  8f 45 8d 40 6a fc a4 26 ae 1e a8 7d 8b 48 88 c2
  20 b3 c3 e3 93 2b 02 03 01 00 01

[06:53:07,581] INFO CGA - Verification of CGA was successful!
[06:53:07,581] INFO  TimeStamp  - Verifying Timestamp
[06:53:07,581] INFO  TimestampCache  - 1.223292216E9 >
1.223292209E9 ?
[06:53:07,581] INFO  RSASignature  - Verifying KeyHash
[06:53:07,582] INFO  RSASignature  - Keyhash was successfully
verified
[06:53:07,582] INFO  RSASignature  - Verifying digital
signature
[06:53:07,585] INFO  RSASignature  - Digital signature
successfully verified

[06:53:07,585] DEBUG Monitor  - Modified Packet

  60 00 00 00 00 20 3a ff fe 80 00 00 00 00 00 00
  2c 8e 4f 5b 08 4b 80 c1 fe 80 00 00 00 00 00 00
  20 ee 39 bb 94 e7 35 6a 87 00 7b a6 00 00 00 00
  fe 80 00 00 00 00 00 00 20 ee 39 bb 94 e7 35 6a
  01 01 00 0c 29 c4 87 be

[06:53:07,585] INFO  Monitor  - Reinjected!!!
```

Fig. 6: Example of logs

Easy-SEND has a powerful event logger to record all the SEND events for troubleshooting and learning purposes. It is based on *Apache log4j*[5], a Java-based logging framework. Our event logger has three messages levels (INFO, DEBUG, and ERROR) that can be activated. INFO messages show information related to the states and processes that occur in the application. DEBUG messages allow users to get a hexadecimal dump of the messages sent or received by *Easy-SEND*. ERROR messages are shown when abnormal situations occur.

[4] http://www.savarese.org/software/vserv-ipq

[5] http://logging.apache.org/log4j/1.2/index.html

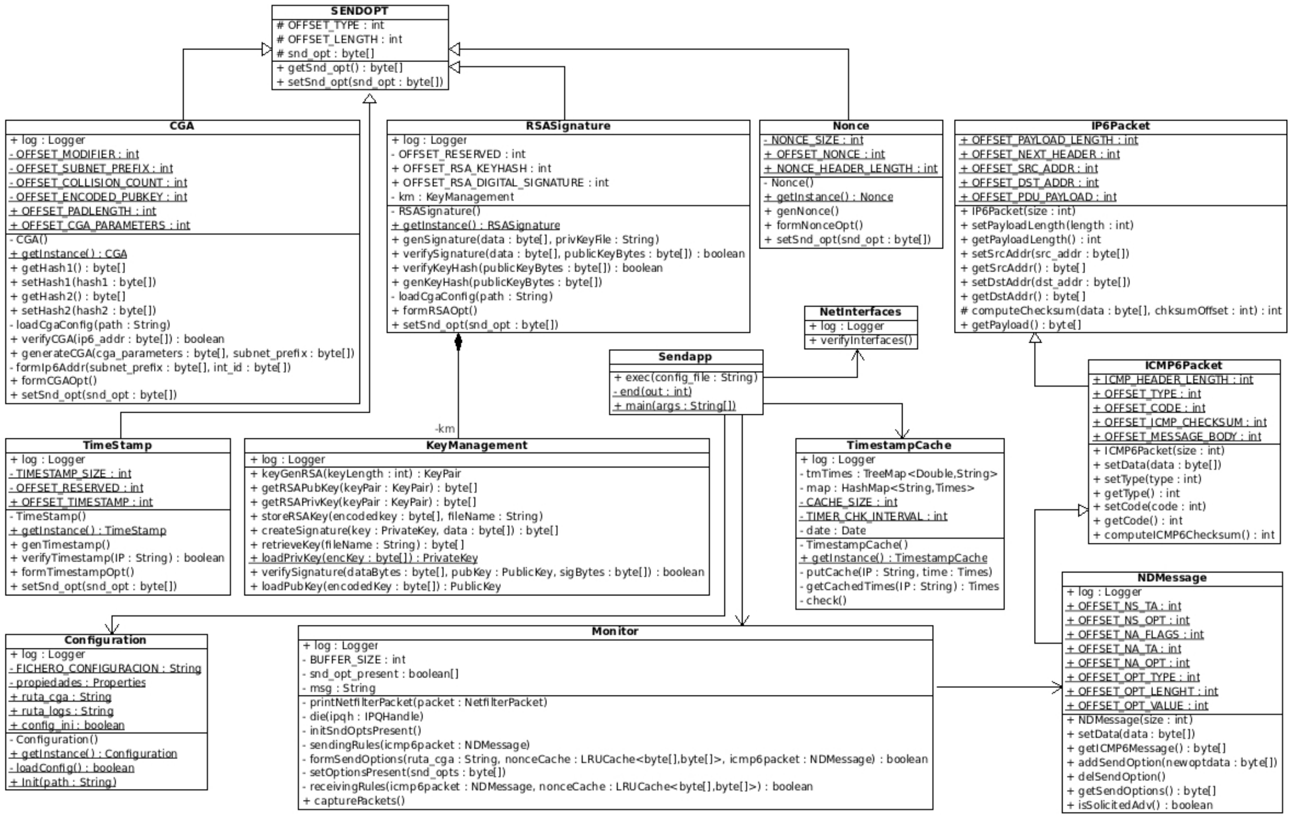
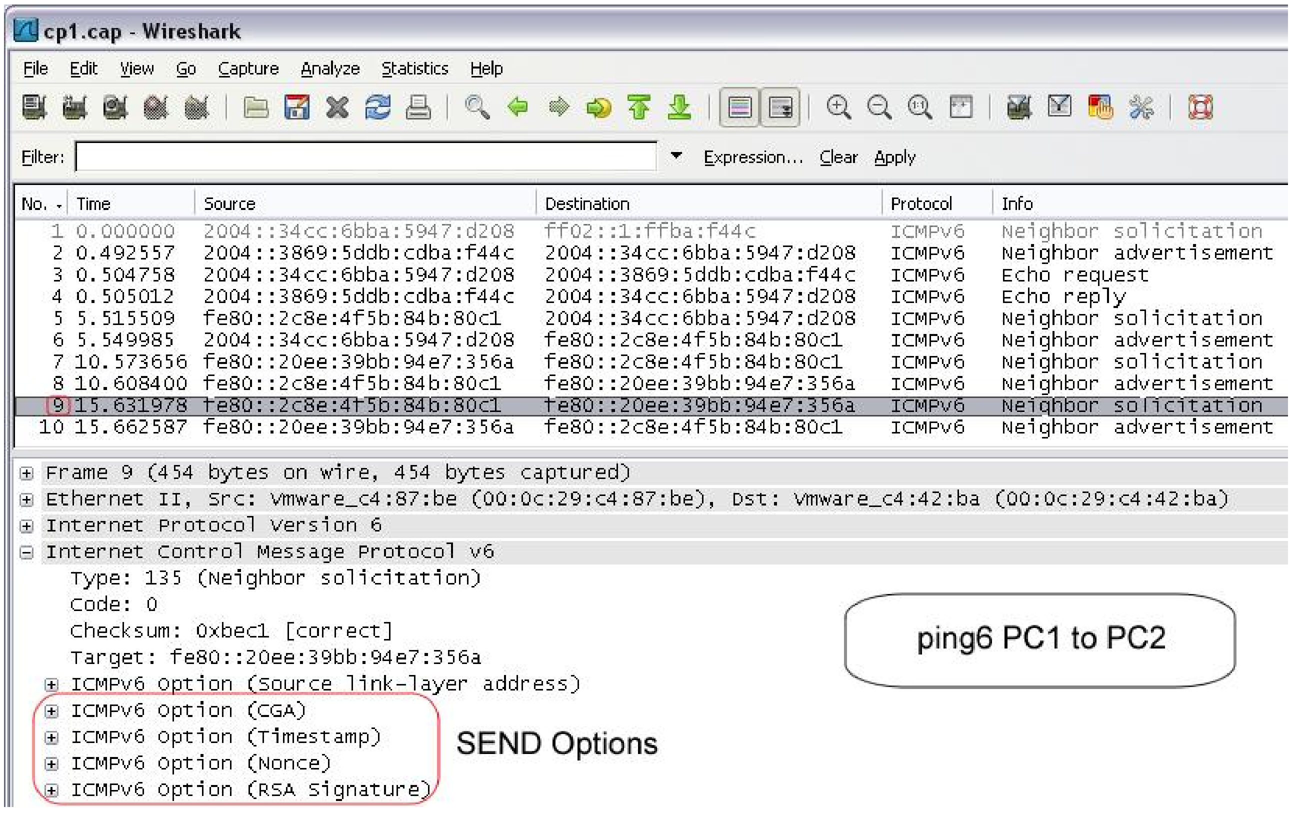

SENDOPT
# OFFSET_TYPE : int
# OFFSET_LENGTH : int
# snd_opt : byte[]
+ getSnd_opt() : byte[]
+ setSnd_opt(snd_opt : byte[])
CGA
+ log : Logger
- OFFSET_MODIFIER : int
- OFFSET_SUBNET_PREFIX : int
- OFFSET_COLLISION_COUNT : int
- OFFSET_ENCODED_PUBKEY : int
+ OFFSET_PADLENGTH : int
+ OFFSET_CGA_PARAMETERS : int
- CGA()
+ getInstance() : CGA
+ getHash1() : byte[]
+ setHash1(hash1 : byte[])
+ getHash2() : byte[]
+ setHash2(hash2 : byte[])
- loadCgaConfig(path : String)
+ verifyCGA(ip6_addr : byte[]) : boolean
+ generateCGA(cga_parameters : byte[], subnet_prefix : byte[])
- formIp6Addr(subnet_prefix : byte[], int_id : byte[ ])
+ formCGAOpt()
+ setSnd_opt(snd_opt : byte[])
RSASignature
+ log : Logger
- OFFSET_RESERVED : int
+ OFFSET_RSA_KEYHASH : int
+ OFFSET_RSA_DIGITAL_SIGNATURE : int
- km : KeyManagement
- RSASignature()
+ getInstance() : RSASignature
+ genSignature(data : byte[], privKeyFile : String)
+ verifySignature(data : byte[], publicKeyBytes : byte[]) : boolean
+ verifyKeyHash(publicKeyBytes : byte[]) : boolean
+ genKeyHash(publicKeyBytes : byte[])
- loadCgaConfig(path : String)
+ formRSAOpt()
+ setSnd_opt(snd_opt : byte[])
Nonce
- NONCE_SIZE : int
+ OFFSET_NONCE : int
+ NONCE_HEADER_LENGTH : int
- Nonce()
+ getInstance() : Nonce
+ genNonce()
+ formNonceOpt()
+ setSnd_opt(snd_opt : byte[])
IP6Packet
+ OFFSET_PAYLOAD_LENGTH : int
+ OFFSET_NEXT_HEADER : int
+ OFFSET_SRC_ADDR : int
+ OFFSET_DST_ADDR : int
+ OFFSET_PDU_PAYLOAD : int
+ IP6Packet(size : int)
+ setPayloadLength(length : int)
+ getPayloadLength() : int
+ setSrcAddr(src_addr : byte[])
+ getSrcAddr() : byte[ ]
+ setDstAddr(dst_addr : byte[])
+ getDstAddr() : byte[ ]
# computeChecksum(data : byte[], chksumOffset : int) : int
+ getPayload() : byte[]
NetInterfaces
+ log : Logger
+ verifyInterfaces()
Sendapp
+ exec(config_file : String)
- end(out : int)
+ main(args : String[])
ICMP6Packet
+ ICMP_HEADER_LENGTH : int
+ OFFSET_TYPE : int
+ OFFSET_CODE : int
+ OFFSET_ICMP_CHECKSUM : int
+ OFFSET_MESSAGE_BODY : int
+ ICMP6Packet(size : int)
+ setData(data : byte[])
+ setType(type : int)
+ getType() : int
+ setCode(code : int)
+ getCode() : int
+ computeICMP6Checksum() : int
-km
TimeStamp
+ log : Logger
- TIMESTAMP_SIZE : int
- OFFSET_RESERVED : int
+ OFFSET_TIMESTAMP : int
- TimeStamp()
+ getInstance() : TimeStamp
+ genTimestamp()
+ verifyTimestamp(IP : String) : boolean
+ formTimestampOpt()
+ setSnd_opt(snd_opt : byte[])
KeyManagement
+ log : Logger
+ keyGenRSA(keyLength : int) : KeyPair
+ getRSAPubKey(keyPair : KeyPair) : byte[]
+ getRSAPrivKey(keyPair : KeyPair) : byte[]
+ storeRSAKey(encodedkey : byte[], fileName : String)
+ createSignature(key : PrivateKey, data : byte[]) : byte[]
+ retrieveKey(fileName : String) : byte[ ]
+ loadPrivKey(encKey : byte[]) : PrivateKey
+ verifySignature(dataBytes : byte[], pubKey : PublicKey, sigBytes : byte[]) : boolean
+ loadPubKey(encodedKey : byte[]) : PublicKey
TimestampCache
+ log : Logger
- tmTimes : TreeMap<Double,String>
- map : HashMap<String,Times>
- CACHE_SIZE : int
- TIMER_CHK_INTERVAL : int
- date : Date
- TimestampCache()
+ getInstance() : TimestampCache
- putCache(IP : String, time : Times)
- getCachedTimes(IP : String) : Times
- check()
NDMessage
+ log : Logger
+ OFFSET_NS_TA : int
+ OFFSET_NS_OPT : int
+ OFFSET_NA_FLAGS : int
+ OFFSET_NA_TA : int
+ OFFSET_NA_OPT : int
+ OFFSET_OPT_TYPE : int
+ OFFSET_OPT_LENGHT : int
+ OFFSET_OPT_VALUE : int
+ NDMessage(size : int)
+ setData(data : byte[])
+ getICMP6Message() : byte[]
+ addSendOption(newoptdata : byte[])
+ delSendOption()
+ getSendOptions() : byte[]
+ isSolicitedAdv() : boolean
Configuration
+ log : Logger
- FICHERO_CONFIGURACION : String
- propiedades : Properties
+ ruta_cga : String
+ ruta_logs : String
+ config_ini : boolean
- Configuration()
+ getInstance() : Configuration
- loadConfig() : boolean
+ Init(path : String)
Monitor
+ log : Logger
- BUFFER_SIZE : int
- snd_opt_present : boolean[]
- msg : String
- printNetfilterPacket(packet : NetfilterPacket)
- die(ipqh : IPQHandle)
- initSndOptsPresent()
- sendingRules(icmp6packet : NDMessage)
- formSendOptions(ruta_cga : String, nonceCache : LRUCache<byte[],byte[]>, icmp6packet : NDMessage) : boolean
- setOptionsPresent(snd_opts : byte[])
- receivingRules(icmp6packet : NDMessage, nonceCache : LRUCache<byte[],byte[]>) : boolean
+ capturePackets()

Fig. 7: Diagram of principal classes


cp1.cap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
ping6 PC1 to PC2
SEND Options

```
No. . Time       Source                    Destination               Protocol  Info
  1 0.000000   2004::34cc:6bba:5947:d208 ff02::1:ffba:f44c         ICMPv6    Neighbor solicitation
  2 0.492557   2004::3869:5ddb:cdba:f44c 2004::34cc:6bba:5947:d208 ICMPv6    Neighbor advertisement
  3 0.504758   2004::34cc:6bba:5947:d208 2004::3869:5ddb:cdba:f44c ICMPv6    Echo request
  4 0.505012   2004::3869:5ddb:cdba:f44c 2004::34cc:6bba:5947:d208 ICMPv6    Echo reply
  5 5.515509   fe80::2c8e:4f5b:84b:80c1  2004::34cc:6bba:5947:d208 ICMPv6    Neighbor solicitation
  6 5.549985   2004::34cc:6bba:5947:d208 fe80::2c8e:4f5b:84b:80c1  ICMPv6    Neighbor advertisement
  7 10.573656  fe80::20ee:39bb:94e7:356a fe80::2c8e:4f5b:84b:80c1  ICMPv6    Neighbor solicitation
  8 10.608400  fe80::2c8e:4f5b:84b:80c1  fe80::20ee:39bb:94e7:356a ICMPv6    Neighbor advertisement
  9 15.631978  fe80::2c8e:4f5b:84b:80c1  fe80::20ee:39bb:94e7:356a ICMPv6    Neighbor solicitation
 10 15.662587  fe80::20ee:39bb:94e7:356a fe80::2c8e:4f5b:84b:80c1  ICMPv6    Neighbor advertisement

⊞ Frame 9 (454 bytes on wire, 454 bytes captured)
⊞ Ethernet II, Src: Vmware_c4:87:be (00:0c:29:c4:87:be), Dst: Vmware_c4:42:ba (00:0c:29:c4:42:ba)
⊞ Internet Protocol Version 6
⊟ Internet Control Message Protocol v6
    Type: 135 (Neighbor solicitation)
    Code: 0
    Checksum: 0xbec1 [correct]
    Target: fe80::20ee:39bb:94e7:356a
  ⊞ ICMPv6 Option (Source link-layer address)
  ⊞ ICMPv6 Option (CGA)
  ⊞ ICMPv6 Option (Timestamp)
  ⊞ ICMPv6 Option (Nonce)
  ⊞ ICMPv6 Option (RSA Signature)
```

Fig. 8: Ping between two PCs that run Easy-SEND

Typically, logs can be displayed in real-time in a Unix shell. It is also possible to redirect the logs to text files for later analysis. Our event logger can show many information to users helping them in the troubleshooting or learning process. Fig. 6 shows the trace generated by packet 9 (a Neighbor Solicitation message) for the capture of Fig. 8.

At the present time, *Easy-SEND* is the unique free and open source implementation of the SEND protocol that is active, since DoCoMo USA Labs is no longer supporting its implementation. The source code of *Easy-SEND* can be downloaded from SourceForge (a famous web-based source code repository) at http://sourceforge.net/projects/easy-send. To facilitate the learning and teaching process, a virtual appliance is also available. A virtual appliance is a minimalist virtual machine image designed to run under some sort of virtualization technology and aimed to eliminate the installation, configuration and maintenance costs. Our virtual appliance can be run with *VMWare Player*[6].

VII. Conclusions and Future Work

Security aspects of IPv6 networks are not fundamentally different from IPv4 networks, therefore, many of the possible attacks to IPv4 protocols can also be done in IPv6 protocols, and the ND protocol is not the exception. SEND has been developed to minimize the attacks that can be done to IPv6 nodes during their auto-configuration process. Unlike IPsec, SEND is more easy and practicable to be implemented. Note that [17] proposes a new protocol called TRDP (Trusted Router Discovery Protocol) to secure the router discovery process. Compared with the ADD (Authorization Delegation Discovery) process introduced in the SEND protocol, TRDP obviates the burdensome work for a host to parse the lengthy certification path, improves efficiency of network communication between the router and hosts during the router authentication process, and also reduces the exposure to attacks on both hosts and the access router.

At *Central University of Venezuela*, SEND is an important topic in the syllabus of the *Advanced Network Protocols* course. Instructors had a hard time teaching SEND, so some laboratories were introduced to clarify the complex processes associated to SEND. Initially, the SEND implementation of DoCoMo USA Labs was used, but this implementation has been discarded since its debugging system has some weaknesses and DoCoMo USA Labs announced that it will no longer maintain the project. *Easy-SEND* was developed to face this situation.

Easy-SEND is a free and open source implementation of the SEND protocol developed under the GNU General Public License in Java. It has a powerful event logger with three levels of messages. It main goal is to be used for teaching and learning purpose, but can also be used in a production environment. Since it is open source, its source code can be enhanced or adapted by other researchers for specific needs.

At the present time, *Easy-SEND* is the unique open source project that implements the SEND protocol that is active. We plan to further develop it. Our goal is to offer the application to other national and international universities to be used in their curriculum for teaching the SEND protocol. As future work, we plan to integrate to our application the process of Router Discovery with the ADD trust mechanism, and extension fields to the CGA parameter data structure to support different hash functions [2]. Also, the implementation of *Easy-SEND* in the Linux user-space can not be considered as a final solution, so we will adapt our application to the kernel to optimize the aggregation and the verification of SEND message options. Finally, we want to port our application to Microsoft Windows operating systems.

[6] http://www.vmware.com/products/player

References

[1] J. Arkko, J. Kempf, B. Zill, and P. Nikander. Secure Neighbor Discovery (SEND). RFC 3971. March, 2005.
[2] M. Bagnulo and J. Arkko. Cryptographically Generated Addresses (CGA) Extension Field Format. RFC 4581. October, 2006.
[3] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972. March, 2005.
[4] J. Charvat. Project Management Methodologies: Selecting, Implementing, and Supporting Methodologies and Processes for Projects. John Wiley & Sons. February, 2003.
[5] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443. March, 2006.
[6] J. Davies. Understanding IPv6. Microsoft Press, Second Edition. January, 2008.
[7] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460. December, 1998.
[8] A. Farrel. The Internet and its Protocols: A Comparative Approach. Morgan Kaufmann, First Edition. May, 2004.
[9] S. Hagen. IPv6 Essentials. O'Reilly, Second Edition. May, 2006.
[10] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280. April, 2002.
[11] S. Kent. IP Authentication Header. RFC 4302. December, 2005.
[12] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303. December, 2005.
[13] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP Version 6 (IPv6). RFC 4861. September, 2007.
[14] P. Nikander, J. Kempf, and E. Nordmark. IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756. May, 2004.
[15] A. Orebaugh, G. Ramirez, and J. Beale. Wireshark & Ethereal Network Protocol Analyzer Toolkit. Syngress. February, 2007.
[16] W. Royce. Managing the Development of Large Software Systems: Concepts and Techniques. In Proceedings of the 9th International Conference on Software Engineering, Monterey, California, United States. March 30-April 2, 1987.
[17] J. Zhang, J. Liu, Z. Xu, J. Li, and X. Ye. TRDP: a Trusted Router Discovery Protocol. In Proceeding of the 7th International Symposium on Communications and Information Technologies, Sydney, Australia. October 17-19, 2007.