Native SeND kernel API for *BSD

Ana Kukec
University of Zagreb
anchie@fer.hr

Bjoern A. Zeeb
The FreeBSD Project
bz@FreeBSD.org

Abstract

In the legacy world of Internet Protocol Version 4 (IPv4), the link layer protocol, the Address Resolution protocol (ARP) is known to be vulnerable to spoofing attacks, but has nevertheless been in use entirely unsecured. The Neighbor Discovery Protocol (NDP), which in the IPv6 world roughly corresponds to IPv4 ARP, is vulnerable to a similar set of threats if not secured. The Secure Neighbor Discovery (SeND) extensions counter security threats to NDP by offering proof of address ownership, message protection, and router authorization. The current lack of robust support for SeND within BSD operating system family and drawbacks in the existing reference SeND implementation limits its deployment. We illustrate the protocol enhancements and their implemenation by rehashing the known problem scenarios with unsecured NDP and providing the short information about SeND. We then describe the design and implementation of a new, BSD licensed, kernel-userspace API for SeND, which mitigates the overhead associated with the reference implementation in FreeBSD, and which aims to improve portability to other BSD-derived operating systems.

1 Introduction

IP version 6 (IPv6) [7] has been designed as the successor to IP version 4 (IPv4). Unlike the common opinion that IPv6 is primarly the solution for the problem of the shortage of public IPv4 addresses, there are many other changes from IPv4 to IPv6 such as the hedaer format simplification, improved support for extensions and options, flow labeling capability, and authentication and privacy capabilities. However, the most significant changes are not in the IP protocol itself, but in the supporting protocols and mechanisms that were developed along with it, for example the ones that are related to the communication between link local devices.

The communication between IPv4 link local devices is supported by two protocols:

1. Address Resolution Protocol (ARP) that determines a host's link layer address [17], and

2. Internet Control Message Protocol version 4 (ICMP) that is a messaging system and an error reporting protocol for the IPv4 network layer [18].

ICMP provides various functionalities through the use of ICMP messages, where two important functionalities for the link local communication are ICMP Router Discovery and ICMP Redirect. ICMP Router Discovery messages [6] deal with the configuration of IP hosts with the IP addresses of neighboring routers, using ICMP Router Advertisement messages and ICMP Router Solicitation messages. Since Router Advertisements are used by routers only to advertise their existance and not their location, there is a separate mechanism that uses ICMP Redirect messages to enable routers to convey the information about the optimal, alternate route to hosts. There is also a certain number of ICMP based algorithms that support the IPv4 communication between link local hosts that are recommended for IPv4, but they are not required and widely adopted. [4] defines some possible approaches to solve Dead Gateway Detection, a scenario in which the IP layer must detect the next-hop gateway failure and choose an alternate gateway, but there is no widely accepted IPv4 suite protocol for it.

Even though previously mentioned features work properly in IPv4, they were developed in an ad hoc manner. They consist of a great number of different protocols, mechanisms, algorithms, and Internet Standards. Both the nowadays Internet use case scenarios and security threat analysis are pointing out their various limitations and the need for the enhancements.

IPv6 Neighbor Discovery Protocol (NDP) [15] is a single protocol that corresponds to the combination of all previously mentioned protocols (ARP, ICMP Router

Discovery, ICMP Redirect, and various recommended ICMP mechanisms). Most of the Neighbor Discovery Protocol functionalities are based on the five ICMPv6 control messages (Router Solicitation and Advertisement, Neighbor Solicitation and Advertisement, and Redirect). Router Solicitation is sent by hosts as the request for Router Advertisement. Router Advertisement is sent by routers periodically or as a response to Router Solicitation, to advertise the link local prefix and other options. Neighbor Solicitation is sent by IPv6 hosts to find out a neighbor's link layer address or to verify that a node is still reachable. Neighbor Advertisement is sent by IPv6 hosts as a response to Neighbor Solicitation or to propagate the link layer address change. Redirect is sent by routers to inform hosts of the better first-hop destination.

Neighbor Discovery Protocol functionalities are classified into two groups: host-host functionalities and host-router functionalities. Host-router functionalities enable the host to locate routers on the link local network (router discovery), to differentiate between the link local network and distant networks (prefix discovery), to find out the parameters of the link local network and neighboring routers (parameter discovery), and to autoconfigure their IPv6 address based on the information provided by a router. Host-host functionalities include the address resolution (ARP functionality in IPv4), the next-hop determination based on the datagram's IP destination address, the determination whether the host is directly reachable (neighbor unreachability detection), and the determination of whether the choosed address already exists in the link local network (duplicate address detection). NDP function that does not belong in neither of two previously mentioned groups is the Redirect function. The Renumbering functionality, a mechanism that takes care of the renumbering based on the Router Advertisement messages containing the prefix, sent in a timely manner. The Renumbering mechanism is derived from the combined use of the neighbor discovery and the address autoconfiguration. The Neighbor Discovery Protocol combines all functionalities of IPv4 supporting protocols for the communication between link local devices, but also provides many enhancements and improvements over the mentioned set of protocols. The typical example of one such enhancement is the Neighbor Unreachability Detection [15] (NUD) that is one of the fundamental Neighbor Discovery Protocol parts. IPv4 Dead Gateway Detection [17] (DGD) is similar to Neighbor Unreachability Detection in IPv6, but addresses just a subset of the problems that Neighbor Unreachability Detection deals with. IPv4 Dead Gateway Detection is a simple fail-over mechanism that changes host's default gateway to the next configured default gateway. There is no possibility to distinguish whether the link local or a remote

Figure 1: Attack on Address Resolution

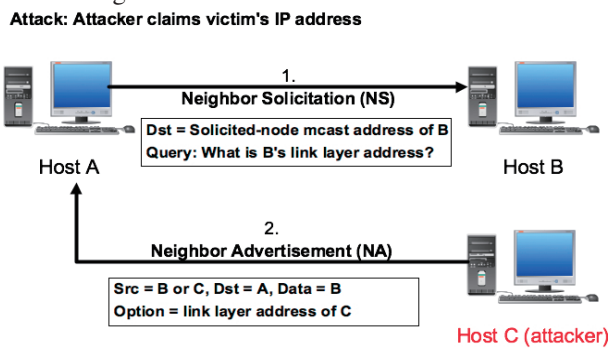
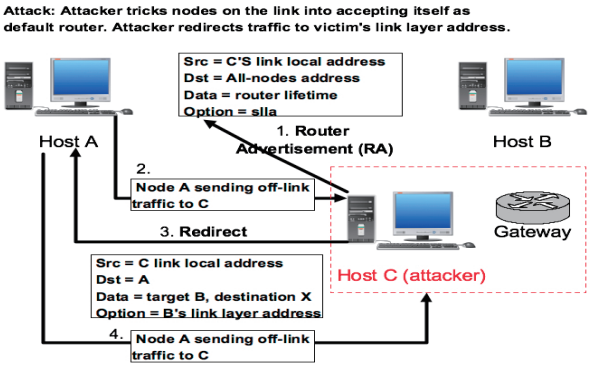


Figure 2: Redirect Attack



gateway has failed, or to get any detailed reachability information. Thus there is no possibility for the fail-back to the previous router. Neighbor Unreachability Detection is enhanced mechanism that allows the node to track the detailed reachability information about its neighbor, either the link local host or the router. Based on the use of ICMPv6 messages, it enables the host to fail-back to the previous router, to make use of the inbound load balancing in case of replicated interfaces, to inform the neighbors about the change of its link layer address.

Both IPv4 protocols supporting the link local communication and the Neighbor Discovery Protocol, if not secured, are vulnerable and affected by the similar set of threats. The initial Neighbor Discovery Protocol specification proposed the use of IPsec, specifically IP Authentication Header (AH) [9] and IP Encapsoluationg Security Payload [10], for the protection, by authentication the packet exchanged to overcome the shortcomings. Unlike the Neighbor Discovery Protocol that can be secured with the Secure Neighbor Discovery (SeND), one of the significant shortcomings of the IPv4 protocols supporting the link local communication, such as the Address Resolution Protocol and other ICMP-based mech-

Figure 3: DAD Attack

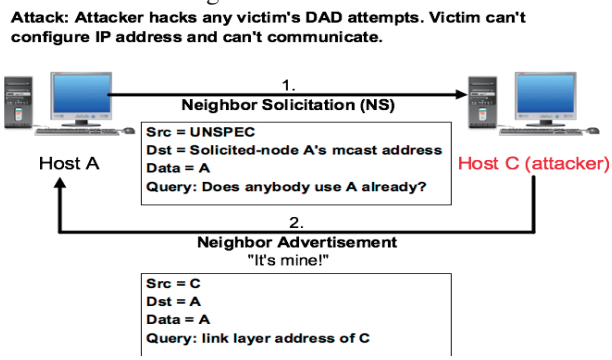
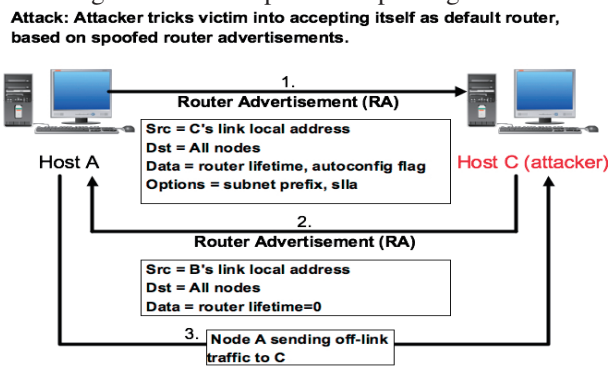


Figure 4: First-Hop Router Spoofing Attack



anisms, is that there is no standardized, widely adopted enhancement for securing them.

The next section discusses the main threats associated to the Neighbor Discovery Protocol, illustrating the real world attacks that have never been solved for IPv4, but are solved for IPv6. It will further explain why the inital proposal for the Neighbor Discovery Protocol protection with IPsec was abandoned in favour of SeND.

2 Background

2.1 Neighbor Discovery Protocol (NDP) threats

The Neighbor Discovery Protocol trust models and threats are well known and clearly described in [16]. It illustrates the following attacks:

- Attack on Address Resolution (Figure 1),
- Redirect Attack (Figure 2),
- Duplicate Address Detection (DAD) Attack (Figure 3),
- First-Hop Router Spoofing Attack (Figure 4),
- Address Configuration Attack (Figure 5).

The Neighbor Discovery Protocol [15] offers some basic protection mechanisms. For example it introduces the limitation for the IPv6 source address to be either the unspecified address (::/128) or a link-local address, or puts the limitation on the hop limit to be set to 255, trying to limit source address spoofing by making sure that packet is coming from a host on a directly connected network. However, the protection shield offered by the Neighbor Discovery Protocol itself is not enough to encounter most of the known threats. This is due the fact that Neighbor Discovery Protocol as it is, is not able to offer any authentication, message protection or router authorization capabilities.

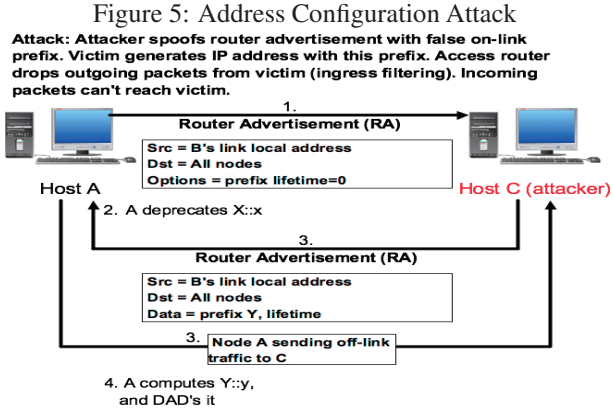
2.2 Neighbor Discovery Protocol and IPsec

The initial Neighbor Discovery Protocol specification proposed the use of IPsec Authentication Header (AH) to encounter known threats. This approach appeared to be problematic. Theoretically, in the IPv6 architecture, it is possible to secure all IP packets, including ICMPv6 and Neighbor Discovery messages, even the ones sent to multicast addresses. Packets that are supposed to be secured are filtered based on the Security Policy Database, and then protected based on Security Associations maintained automatically by the Internet Key Exchange protocol (IKE). But here we end up with the chicken-and-egg bootstraping problem [1]. IKE is not able to establish a Security Association between the local hosts because in order to send the IKE UDP message it would have had to send the Neighbor Solicitation message, which would have required the Security Association which does not exist. Even if we decide to use a manual configuration for Security Associations, which solves the bootstrapping problem, we would be faced with the problem of maintaining an enormous number of Security Associations, especially when considering multicast links (Neighbor Discovery and Address Autconfiguration use a few fixed multicast addresses plus a range of 16 million ”solicited node” multicast addresses). Even in scenarios with only a small fraction of the theoretically maximum number of addresses, which appear to be very common in case of the local communication, statically preconfigured Security Associations make the use of IPsec impractical.

2.3 Secure Neighbor Discovery (SeND)

Neighbor Discovery needed a different approach to encounter threats, a cryptographic extension to the basic protocol that will not require the excessive manual keying. To solve the problem IETF SeND working group that was chartered in 2002 defined the initial SeND spec-

Figure 5: Address Configuration Attack



ification which was recently updated by RFC4861 [15]. The important thing to notice is that Secure Neighbor Discovery is not a new protocol, but just a set of enhancements to the Neighbor Discovery Protocol. It is based on four new Neighbor Discovery options prepending the normal Neighbor Discovery message options, and two new messages.

Secure Neighbor Discovery enhances the Neighbor Discovery Protocol with the following three additional features:

1. address ownership proof,
2. message protection,
3. router authorization.

The addres ownership proof prevents the attacker from stealing the IPv6 address, which is a fundamental problem for the router discovery, duplicate address detection and address resolution mechanisms. This feature is based on IPv6 addresses known as Cryptographically Generated Addresses (CGAs). CGA is a mechanism that binds the public component of a public-private key pair to an IPv6 address. It is generated as a one-way hash of the four input values: a 64-bit subnet prefix, the public key of the address owner, the security parameter (sec) and a random nonce (modifier).

$$CGA(128) = Prefix(64)|IID(64)$$

$$IID(64) = hash(prefix, pubkey, sec, modifier)$$

The detailed description of the CGA generation procedure is described in RFC3972 [2].

The owner of the CGA address sends the all CGA Parameters, including all required input data for the CGA generation together with the CGA address to the verifier. The CGA verification consists of the re-computation and comparison of received CGA value based on the received CGA parameters, including the public key. However, the hash of the public key itself offers no protection at all, if it is not used in combination with the digital signature produced using the corresponding private key. When using CGAs in Secure Neighbor Discovery, the sender signs the message with the private key that is possessed only by him, and that is the key related to the public key used in CGA's interface identifier generation. This prevents an attacker from spoofing a cryptographically generated address. All the information about the CGA parameters, such as the public key used for the CGA verification, are exchanged within the new Neighbor Discovery Protocol option - the CGA option. The impact of the collision attacks in CGAs is described in RFC4982 [3]. Attacks against the collision-free property of hashes are known, but their characteristic is that they deal with the non-repudiation features. The attacker would be able to create two different messages that result in the same hash, and then use them interchangeably. The important thing to notice is that both messages must be produced by the attacker. Since the usage of CGAs in SeND does not include the provision of the non-repudiation capabilities, it is not affected by the hash collision attacks.

SeND offers message protection in terms of the message integrity protection of all messages relating to neighbor and router discovery, using the new Neighbor Discovery option called RSA option. It contains a public key digital signature calculated over the message, and thus protects the integrity of the message and authenticates the identity of the sender. Secure Neighbor Discovery message that the sender signs with its private key includes the link layer information, which creates the secure binding between the IP address and link layer anchor. In such a way, Secure Neighbor Discovery allows for the verification with the signer's public key that the host's IP address is bound to the trustworthy lower layer anchor. The public key trust is achieved either through the CGA address ownership proof (in the neighbor discovery procedure), or through the X.509 certificate chain (in the router discovery procedure), or both. SeND also defines the Timestamp and Nonce options to protect messages from reply attacks, and to ensure the request/response correlation.

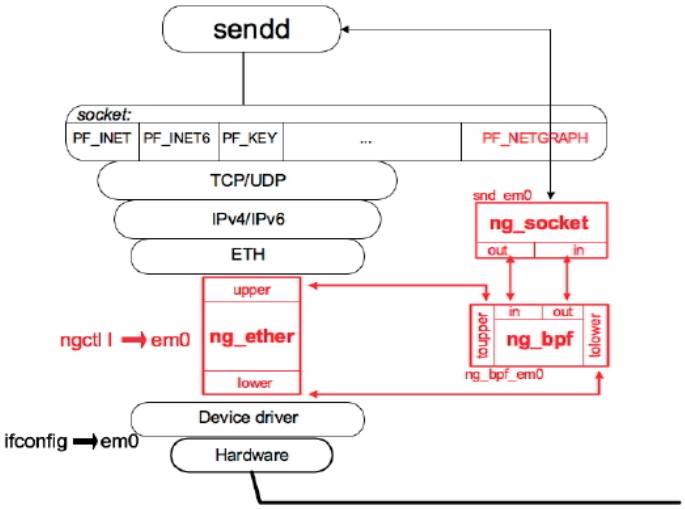
The router authorization feature introducies two novelties to Neighbor Discovery:

1. it authorizes routers to act as default gateways for a certain local network, and
2. specifies prefixes that an authorized router may advertise on this certain link.

A new host on the link can easily configure itself using the information learned by the router, while in the same time there is no way a host can tell from the Neighbor Discovery information, that the router is actually an

authorized router. If the link is unsecured, the router might be a rogue router. At the moment when the host should verify whether the router is a valid one, the host is not able to do so since it is not able to communicate with the off-link hosts. To solve this situation, SeND introduces two new messages: Certification Path Solicitation message and Certification Path Advertisement message. The first one is sent by newly connected host to the router. The second one is the response sent by the router, and contains the certificate chain that contains the certification path, that the host uses to validate the router. The certificate path consists of the Router Authorization Certificate that authorizes a specific IPv6 node to act as a router, followed by intermediate certificates that lead to the trust anchor trusted both by the router and the host. Trust between the router and the hosts is achieved through the third party - the trust anchor (X.509 Certification Authority) [5]. The Router Authorization Certificate contains the information about the prefix that he is authorized to advertize.

Figure 6: NTT DoCoMo's send02 using netgraph nodes and BPF



3 Implementation

Neighbor Discovery Protocol is widely supported by many modern operating systems, since the NDP support is mandatory for IPv6 network stacks. The code resides mainly in kernel. However, there are very few Secure Neighbor Discovery implementations. None of the contemporary open source operating systems ships with built-in support for SeND.

3.1 Existing SeND implementations

The open-source SeND reference implementation (send-0.2), originally developed by NTT DoCoMo, works on Linux and FreeBSD. On FreeBSD, this implementation uses a Berkley Packet Filter (BPF) interface embedded in a netgraph node (ng_bpf) to divert SeND traffic from kernel to an userland daemon, and vice versa. This approach has two major drawbacks. First, all network traffic (both SeND and non-SeND) has to traverse through a ng_bpf filtering node (and through the netgraph subsystem in general), which introduces significant processing overhead, effectively prohibiting production deployment of SeND in high-speed networking environments. And second, the current send-0.2 implementation depends on the netgraph subsytem, which is available only in FreeBSD and DragonFlyBSD, making in send-0.2 implementation being unusable on other BSD-derived operating systems, such as NetBSD, OpenBSD or Mac OS X.

Figure 6 illustrates the design of DoCoMo's SeND implementation for FreeBSD. The communication between the Neighbor Discovery stack implemented in kernel and the Secure Neighbor Discovery daemon flows through the chain of netgraph nodes: ng_ether, ng_bpf and ng_socket. Packets that are incoming from the interface's point of view are protected with Secure Neighbor Discovery options (CGA option, RSA Signature option, Timestamp and Nonce option). Before the kernel will be able to process them in its Neighbor Discovery stack the packetmust be validated and the Secure Neighbor Discovery options which are all unkown to kernel must be stripped of. Initially, all incoming packets arrive to the ng_ether "lower" hook, which is a connection to the raw Ethernet device and from there on to the ng_bpf "tolower" hook. That netgraph node will filter out the incoming packets that are protected by SeND options and pass these packets through an ng_socket "out" hook to the SeND daemon in user space, rather than passing them on inside the kernel for normal upper layer processing.

In userland, Secure Neighbor Discovery options are checked. Upon sucessfull validation all Secure Neighbor Discovery options are removed, and injected back to kernel, through the ng_socket "out" hook, ng_bpf "toupper" hook and ng_ether "upper" hook, is a pure Neighbor Discovery message. The kernel will then pass the packets on through the normal input path to the upper layers and process the Neighbor Discovery information. In case that daemon cannot successfully validate the SeND options, it will silently drop the packet.

Packets that are outgoing from the interface's point of view must be sent to Secure Neighbor Discovery daemon just before they are supposed to exit the outgoing interface. After the kernel upper layer processing, which includes the Neighbor Discovery stack processing, all outgoing packets are forwarded throughthe ng_ether "upper"

hook to the ng_bpf node. They are injected to the userland where the Secure Neighbor Discovery adds additional options to protect the packet. Packets on that way flow through ng_bpf "out" hook and ng_socket "in" hook to the userland. The Secure Neighbor Discovery daemon prepends the normal Neighbor Discovery options in the packet with the CGA option, RSA Signature option, Timestamp and Nonce option, and sends the packet back to kernel through the ng_socket in hook and the ng_bpf tolower hook to ng_ether. Packets then leaves the interface through the ng_ether lower hook, which is the direct connection to the lower device link layer.

As mentioned previously, Secure Neighbor Discovery also enhances the Neighbor Discovery Protocol with two new messages that participate in the process of router authorization. Neither the Certification Path Solicitation message, nor the Certification Path Advertisement message are processed in Neighbor Discovery kernel stack since they are not the part of the basic Neighbor Discovery Protocol. Thus both new messages are not exchanged through netgraph nodes, but through the separate socket.

While the NTT DoCoMo implementation had the advantage, that it was written to be distributed independetly of the operating system, not needing any operating system changes, it had the drawbacks of using the netgraph subsystem as well as hitting the Berkeley Packet Filter for every packet. To address those problems the operating system itself has to be extended and the following sections will discuss those changes.

3.2 Initial design decisions

- Avoid the use of netgraph.
  Netgraph itself introduces the big overhead to processing. Secondly, as the netgraph subsystem is not available throughout the entire BSD operating system family, it was not considered to be an option for a portable implementantion. Further, avoiding the need of netgraphm, could make an implementation even more portable to other Unices as well.

- Avoid the use of BPF.
  Using the Berkeley Packet Filter meant that all packets, forwarded, for the local system or locally originated would be affected and that this would reduce the performance of a lot of systems, especially if connected to high speed networks, processing lots of packets per second.

- Only defer processing of packets that might be affeccted by Secure Neighbor Discovery.
  As only few ICMPv6 Neighbor Discovery packets are actually affected by SeND it was clear that we should only actually defer processing of those few packets, rather than all. We would also never be interested in packets, that were invalid at a certain (lower) layer. Letting the already existing kernel code do those checks and the handling for us, would mitigate the risk of possible exploits through crafted packets outside the core problem domain of Secure Neighbor Discovery.

- Trigger only on the Secure Neighbor Discovery in case SeND code was loaded.
  Using kernel hooks that will not fire unless the send.ko kernel module was loaded would ensure that normal Neighbor Discovery processing would not be affected for the default case. In case the kernel module would be loaded it would guarantee that all messages would traverse properly through the Neighbor Discovery stack, as if it there was no SeND daemon invloved in the processing.

- Use routing control sockets.
  The routing control sockets have been chosen for their simplicity to exchange messages between kernel and userland, as they are easy to extend beyond the scope of pure routing messages. Actually this had been done before by the net80211 stack. Alternatives would have been to introduce a new, private interface or extend another existing one, like the PF_KEY Key Management API [14], which would have been way more complex.

- Add as few new code to the kernel as possible.
  It was clear that changes to the kernel should be kept to a minimum to ease portability and review, as well as reducing the risk of introducing problems complicating normal processing paths.

- Keep the separate socket to exchange Certification Path Solicitations and Certification Path Advertisements.
  Since those options are exchanged end-to-end between Secure Neighbor Discovery daemons without the use of the Neighbor Discovery kernel code, there is no need to modify the kernel for those but entirely keep their processing in user space.

- Keep the user space implementation.
  If possible and to not re-invent the wheel of handling the configuration and the actual processing of the SeND payload, the NTT DoCoMo SeND daemon should be kept but modified for the new kernel-userland API. This would further allow already existing users to update without the need for changes in their deployment (apart from kernel and daemon updates).

Figure 7: Incoming Neighbor Discovery packet from the wire.

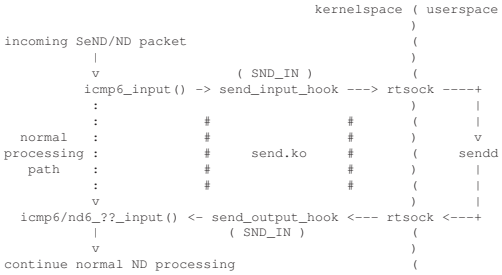
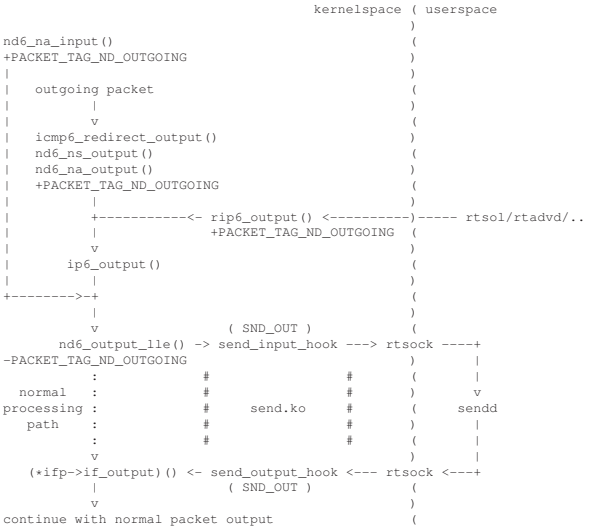


Figure 8: Outgoing Neighbor Discovery packet (reply or locally triggered)



3.3 Native SeND kernel API for *BSD

The goal for the changes were to design and implement a new kernel-userspace API for SeND mitigates the overhead associated with netgraph and BPF and would be easily portable.

In order to accomplish the implementation of such an API, we separated the kernel changes into three main parts:

1. Processing hooks to the existing Neighbor Discovery (ND) input and output code.
2. The SeND kernel module for the dispatching logic.
3. Extensions to the routing control sockets for the SeND kernel-userland interface.

The basic code flow is as follows: incoming Neighbor Discovery packets or outgoing Secure Neighbor Discovery packets are sent to the userland through the send input hook. Neighbor Discovery packets are then passed through the routing socket to the Secure Neighbor Discovery daemon either for protection validation (incoming packets) or (outgoing packets). On the way back to kernel, packets traverse again through the routing socket, but then through the send output hook. While the incoming packets are sent back to Neighbor Discovery stack in kernel, outgoing packets are then sent from the output hook to if_output() routines.

In the following sections we will describe the individual changes for each part in more detail.

The changes to the IPv6 part of the network stack can be separated into Neighbor Discovery input and output path.

For the input path changes were mainly to the `icmp6_input()` function. There we have to divert the ND packet for the following ICMPv6 types: `ND_ROUTER_SOLICIT`, `ND_ROUTER_ADVERT`, `ND_NEIGHBOR_SOLICIT`, `ND_NEIGHBOR_ADVERT` and `ND_REDIRECT`. Instead of directly calling the respective function for direct processing of those ND types, we first check if the `send_input_hook` and with that SeND processing is enabled. If it is we pass the packet to the send.ko kernel module for dispatching to user space. If SeND processing is not enabled, the packet will follow the standard code path to the normal ND handler function.

Pseudo-Code:

```
...
case ND_?????:
  ...
  /*
   * Send incoming SeND-protected/ND
   * packet to user space.
   */
  if (send_input_hook != NULL) {
       send_input_hook(m, ifp, SND_IN,
   ip6len);
       return (IPPROTO_DONE);
  }
  nd6_??_input(m, off, icmp6len);
  ...
  break;
...
```

For the output paths the changes are a bit more diverse and complicated. This is because we can have three different ways that outgoing Neighbor Discovery packets can be send:

1. via `nd6_na_input()` when flushing the ”hold queue” (a list of packets that could not be sent out because of the formerly missing link layer information of the next-hop) in repsonse to the newly learned link layer information.

2. by `icmp6_redirect_output()` function, `nd6_ns_output()`, or `nd6_na_output()`,
3. or from user space applications like *rtsol(8)* or *rtadvd(8)* via `rip6_output()`.

None of those functions directly outputs the packet and as we need to know the IPv6 header for the address, we have to postpone SeND processing to a later point in the output path. To be able to identify the packets later though, we add an attribute, a "tag", to the *mbuf(9)* in the formerly mentioned functions, if SeND processing is enabled. We also save the type as meta-information along the way, though you may find that this will only be used for assert.

Pseudo-Code:

```
...
struct m_tag *mtag;

if (send_input_hook != NULL) {
    mtag = m_tag_get(
        PACKET_TAG_ND_OUTGOING,
        sizeof(unsigned short),
        M_NOWAIT);
    if (mtag == NULL)
            goto fail;
    *(unsigned short *)(mtag + 1) =
        nd->nd_type;
    m_tag_prepend(m, mtag);
}
...
```

As you might notice, there is a slight difference in processing the outgoing Neighbor Solicitation, Neighbor Advertisement and Redirect messages compared to the processing of Router Solicitation and Router Advertisement messages.

Neighbor Solicitations, Neighbor Advertisements and Redirects are handled fully in the Neighbor Discovery kernel stack. Generated messages are tagged with the m_tag `PACKET_TAG_ND_OUTGOING` right after they are recognized in the Neighbor Discovery kernel stack to be the output messages. This happens in sys/netinet6/nd6_nbr.c in the `nd6_ns_output()` and the `nd6_na_output()` functions, as well as and icmp6.c in `icmp6_redirect_output()`.

The difference with outgoing Router Solicitation and Router Advertisement messages is, that they are generated by rtsol and rtadvd daemons and not with the kernel itself. Because of that, we cannot easily tag a packet. We solved this problem by using the already available socket, packet type and ICMPv6 informations in `rip6_output()` in sys/netinet6/raw_ip6.c and conditionally tagging those packets there as well.

Pseudo-Code:

```
...
/*
 * Tag RA/RS messages from rtadvd/rtsol
 * to be sent to
 * user land for SeND protection later.
 */
if (send_input_hook != NULL &&
  so->so_proto->pr_protocol ==
      IPPROTO_ICMPV6) {
    switch (icmpv6_type) {
    case ND_ROUTER_ADVERT:
    case ND_ROUTER_SOLICIT:
        mtag = m_tag_get(
  PACKET_TAG_ND_OUTGOING,
          sizeof(unsigned short),
          M_NOWAIT);
        if (mtag == NULL)
            goto bad;
        m_tag_prepend(m, mtag);
    }
}
...
```

Our tests showed that neither rtadvd nor rtsol, or any other third part user space application sending RA or RS messages needs to be modified for SeND processing, as that is handled transparently for them, with only minimal changes to the kernel.

Depending on the code path, packets will be passed on to `ip6_output()`, which will amogst other things add the IPv6 header and `nd6_output_lle()`, which would pass the packet to the interface's output queue. Prior this step, we check if the packet was previously tagged by us and defer it for output path SeND processing (Figure 8).

Pseudo-Code:

```
...
/*
 * Send outgoing NS/NA/REDIRECT packet
 * to sendd.
 */
if (send_input_hook != NULL) {
   mtag = m_tag_find(m,
       PACKET_TAG_ND_OUTGOING, NULL);
   if (mtag != NULL) {
       send_input_hook(m, ifp,
   SND_OUT, ip6len);
       return;
   }
   ...
}
...
```

The send.ko kernel module consists of three things: the `send_input_hook` and the `send_output_hook`, as well as the module handling logic that also takes care of enabling or disabling the hooks upon load and unload.

The input and output hooks are named after the direction between kernel and userland. It should not be confused with the incoming and outgoing direction of the Neighbor Discovery packets.

- The `send_input_hook` takes packets from the IPv6 network stack's input and output paths and passes them on to the kernel-userland interface for processing by the Secure Neighbor Discovery daemon.
- The `send_output_hook` gets packets from the userland-kernel interface after processing by the Secure Neighbor Discovery daemon to re-inject the packets back into the IPv6 network stack.

In addition both hooks take an argument that describes the direction of the packet:

- `SND_IN` is used for packets originated in the IPv6 input path. These packets are usually protected by Secure Neighbor Discovery options and are sent to userland first via the `send_input_hook` to be validatied and all additional options to be stripped off. When the packets are sent back again to kernel for further Neighbor Discovery kernel stack processing they are still tagged with `SND_IN` even though they pass the `send_output_hook` (Figure 7).
- `SND_OUT` describes both reply or locally originated outgoing packets. These pure Neighbor Discovery packets, are sent to userland to be protected with the Secure Neighbor Discovery options, after the normal processing in the Neighbor Discovery kernel stack via the `send_input_hook`. Once userspace is done, they are sent back to kernel via the `send_output_hook` to be sent out of the interface using the standard output routines (Figure 8).

The last changes needed to the kernel were to interact with userspace. The routing control sockets interface was chosen for its simplicity and flexibility to be extended.

Messages between the Neighbor Discovery kernel stack and send.ko module and the Secure Neighbor Discovery daemon are exchanged through the routing socket.

The routing message type for the rt_msghdr structure of the routing message indicating the Secure Neighbor Discovery event is `RTM_SND` and is defined in sys/net/route.h. The rtm_seq field of the routing message, which is by sender to identify the action is set to either `RTM_SND_IN` or `RTM_SND_OUT`. This is done in parallel to `SND_IN` or `SND_OUT` indicating either the incoming or outgoing direction of messages that are passing through the routing socket. Again the direction is independent from the send.ko module input or output hook naming.

The `rt_securendmsg()` function in sys/net/rtsock.c handles the generation of the routing socket message indicating the Secure Neighbor Discovery event, and it preserves all the existing functions, i.e. for appending the Neighbor Discovery or Secure Neighbor Discovery data to the routing message header. The same had been done before for the net80211 stack with `rt_ieee80211msg()`.

Input from userland back to the kernel is handled by extending the `route_output()` function. The rt_msghdr is stripped of, and the packet is passed to the send.ko `send_output_hook` again.

4 Future work

The decision to use the routing control socket for the interaction with the userspace was made to overcome complexities that would appear in case of the alternative approaches. However, there is the drawback caused by this design decision, due to the unability of the routing socket to provide better control of related daemon. First step to improve our solution is to replace the routing socket in order to provide the appropriate control over the active daemon and a default policy in case of no active daemon in the user spacee.

Along with the development of the native kernel API for SeND, we have continued the development of a Secure Neighbor Discovery userspace application. The current implementation is still based on NTT DoCoMo's initial send-0.2 version. See the availability section for where to find our version of the SeND userspace implementation. Future steps in the development of the user space application will include the implementation of the new Secure Neighbor Discovery specifications that have been developed in the IETF Certificate and Send maIntainance (CSI) working group. They are related to the DHCPv6 and CGA interaction [8], the support of the hash agility in Secure Neighbor Discovery [13], the support of proxy Neighbor Discovery for Secure Neighbor Discovery [12] and the certificate management in the authorization delegation discovery process [11].

5 Conclusion

This paper reasons the need for the Secure Neighbor Discovery extension to counter threats to the Neighbor Discovery Protocol by illustrating the set of security threats, the protocol enhancements that counter those threats and their implementation. It also describes our implementation of a native kernel-userspace SeND API for *BSD

operating systems.

Our prototype is compliant with the Secure Neighbor Discovery specification, both in case of host-host scenarios and router-host scenarios. In case that send.ko module is not loaded, kernel operates just as there was no additional Secure Neighbor Discovery code involved. We successfully overcame major drawbacks of the existing SeND implementation for FreeBSD by eliminating the use of netgraph and Berkley Packet Filter. Our code does not affect other ICMPv6 or IPv6 packets in any way. We developed effective and portable solution for Secure Neighbor Discovery, while introducing as few new code to kernel stack as possible.

As the send.ko kernel module acts as a gateway between the network stack and the userland interface, it will be easy to adopt the user space interface to something more fitting without the need to change the kernel network stack again.

The current kernel work is available in the FreeBSD Perforce depot in the `//depot/projects/soc2009/anchie_send/...` branch. The Secure Neighbor Discovery application is available at `http://google-summer-of-code-2009-freebsd.googlecode.com/files/ Ana Kukec.tar.gz`.

6 Acknowledgements

I would like to thank Google for supporting the implementation work and the AsiaBSDCon for supporting this publication. Also, I would like to thank to Marko Zec for the review and inspiry suggestions on both the implemention and the paper work.

References

[1] ARKKO, J. Effects of ICMPv6 on IKE. IETF Draft, Mar. 2003.

[2] AURA, T. Cryptographically Generated Addresses. RFC 3972, Mar. 2005.

[3] BAGNULO, M., AND ARKKO, J. Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs). RFC 3972, July 2007.

[4] BRADEN, T. Requirements for Internet Hosts – Communication Layers. RFC 1122, Oct. 1989.

[5] COOPER, D., FARRELL, S., BOEYEN, S., HOUSLEY, R., AND POLK, W. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List. RFC 5280, Mar. 2005.

[6] DEERING, S. ICMP Router Discovery Messages. RFC 826, Nov. 1982.

[7] DEERING, S., AND HINDEN, R. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, Apr. 2006.

[8] JIANG, S., SHEN, S., AND CHOWN, T. DHCPv6 and CGA Interaction. IETF Draft, Dec. 2009.

[9] KENT, S. IP Authentication Header. RFC 4302, Sept. 2005.

[10] KENT, S. IP Encapsulating Security Payload (ESP). RFC 4303, Sept. 2005.

[11] KRISHNAN, S., KUKEC, A., AND GAGLIANO, R. Certificate profile and certificate management for SEND. IETF Draft, Dec. 2009.

[12] KRISHNAN, S., LAGANIER, J., AND BONOLA, M. Secure Proxy ND Support for SEND. IETF Draft, July 2009.

[13] KUKEC, A., KRISHNAN, S., AND JIANG, S. SeND Hash Threat Analysis. IETF Draft, July 2009.

[14] MCDONALD, D., METZ, C., AND PHAN, B. PF KEY Key Management API, Version 2. RFC 2367, July 2367.

[15] NARTEN, T., NORDMARK, E., SIMPSON, W., AND SOLIMAN, H. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, Sept. 2007.

[16] P. NIKANDER, J. KEMPF, E. N. IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756, May 2004.

[17] PLUMMER, D. RFC 826 - Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware . RFC 826, Nov. 1982.

[18] POSTEL, J. RFC792 - Internet Control Message Protocol. RFC 792, Sept. 1981.