Behave WG
Internet-Draft
Obsoletes: 3338, 2767
(if approved)
Intended status: Standards Track
Expires: February 11, 2012
August 10, 2011

B. Huang
H. Deng
China Mobile
T. Savolainen
Nokia

Dual Stack Hosts Using "Bump-in-the-Host" (BIH)
draft-ietf-behave-v4v6-bih-06

Abstract

Bump-In-the-Host (BIH) is a host-based IPv4 to IPv6 protocol translation mechanism that allows a class of IPv4-only applications that work through NATs to communicate with IPv6-only peers. The host on which applications are running may be connected to IPv6-only or dual-stack access networks. BIH hides IPv6 and makes the IPv4-only applications think they are talking with IPv4 peers by local synthesis of IPv4 addresses. This draft obsoletes RFC 2767 and RFC 3338.

Status of this Memo

This Internet-Draft will expire on February 11, 2012.

Copyright Notice

Table of Contents

1. Introduction

This document describes Bump-in-the-Host (BIH), a successor and combination of the Bump-in-the-Stack (BIS)[RFC2767] and Bump-in-the-API (BIA) [RFC3338] technologies, which enable IPv4-only legacy applications to communicate with IPv6-only servers by synthesizing IPv4 addresses from AAAA records. Section 8 describes the reasons for updating RFC2767 and RFC3338.

The supported class of applications includes those that use DNS for IP address resolution and that do not embed IP address literals in protocol payloads. This includes legacy client-server applications using the DNS that are agnostic to the IP address family used by the destination and that are able to do NAT traversal. The synthetic IPv4 addresses shown to applications are taken from the RFC1918 private address pool in order to ensure that possible NAT traversal techniques will be initiated.

IETF recommends using dual-stack or tunneling based solutions for IPv6 transition and specifically recommends against deployments utilizing double protocol translation. Use of BIH together with a NAT64 is NOT RECOMMENDED [RFC6180].

BIH includes two major implementation options: a protocol translator between the IPv4 and the IPv6 stacks of a host, or an API translator between the IPv4 socket API module and the TCP/IP module. Essentially, IPv4 is translated into IPv6 at the socket API layer or at the IP layer.

When BIH is implemented at the socket API layer, the translator intercepts IPv4 socket API function calls and invokes corresponding IPv6 socket API function calls to communicate with IPv6 hosts.

When BIH is implemented at the networking layer the IPv4 packets are intercepted and converted to IPv6 using the IP conversion mechanism defined in Stateless IP/ICMP Translation Algorithm (SIIT) [RFC6145]. The protocol translation has the same benefits and drawbacks as SIIT.

The location of the BIH refers to the location of the protocol translation function. The location of DNS synthesis is orthogonal to the location of protocol translation, and may or may not happen at the same level.

BIH can be used whenever an IPv4-only application needs to communicate with an IPv6-only server, independently of the address families supported by the access network. Hence the access network can be IPv6-only or dual-stack capable.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] .

This document uses terms defined in [RFC2460] and [RFC4213].

1.1. Acknowledgement of previous work

This document is a direct update to and directly derivative from Kazuaki TSHUCHIYA, Hidemitsu HIGUCHI, and Yoshifumi ATARASHI's Bump-in-the-Stack [RFC2767] and from Seungyun Lee, Myung-Ki Shin, Yong-Jin Kim, Alain Durand, and Erik Nordmark's Bump-in-the-API [RFC3338], which similarly provide a dual stack host means to communicate with other IPv6 hosts using existing IPv4 applications. Section 7 covers the changes since those documents.

2. Components of the Bump-in-the-Host

Figure 1 shows the architecture of a host in which BIH is implemented as a socket API layer translator, i.e., as a "Bump-in-the-API".

```
                  +----------------------------------------------+
                  | +------------------------------------------+ |
                  | |                                          | |
                  | |             IPv4 applications            | |
                  | |                                          | |
                  | +------------------------------------------+ |
                  | +------------------------------------------+ |
                  | |           Socket API (IPv4, IPv6)        | |
                  | +------------------------------------------+ |
```

```
                  | +-[ API translator]-------------------------+ |
                  | | +-----------+ +---------+ +------------+ | |
                  | | | Ext. Name | | Address | | Function   | | |
                  | | | Resolver  | | Mapper  | | Mapper     | | |
                  | | +-----------+ +---------+ +------------+ | |
                  | +------------------------------------------+ |
                  | +--------------------+ +-------------------+ |
                  | |                    | |                   | |
                  | |    TCP(UDP)/IPv4   | |   TCP(UDP)/IPv6   | |
                  | |                    | |                   | |
                  | +--------------------+ +-------------------+ |
                  +----------------------------------------------+
```

Figure 1: Architecture of a dual stack host using protocol translation at socket layer

Figure 2 shows the architecture of a host in which BIH is implemented as a network layer translator, i.e., a "Bump-in-the-Stack".

```
      +------------------------------------------------------------+
      |  +------------------------------------------+              |
      |  |    IPv4 applications                     |              |
      |  |    Host's main DNS resolver              |              |
      |  +------------------------------------------+              |
      |  +------------------------------------------+              |
      |  |    TCP/UDP                               |              |
      |  +------------------------------------------+              |
```

```
      |  +------------------------------------------+ +---------+  |
      |  |    IPv4                                  | |         |  |
      |  +------------------------------------------+ | Address |  |
      |  +------------------+ +---------------------+ | Mapper  |  |
      |  |    Protocol      | |   Extension Name    | |         |  |
      |  |    Translator    | |   Resolver          | |         |  |
      |  +------------------+ +---------------------+ |         |  |
      |  +------------------------------------------+ |         |  |
      |  |    IPv4 / IPv6                           | |         |  |
      |  +------------------------------------------+ +---------+  |
      +------------------------------------------------------------+
```

Figure 2: Architecture of a dual-stack host using protocol translation at the network layer

Dual stack hosts defined in RFC 4213 [RFC4213] need applications, TCP/IP modules and addresses for both IPv4 and IPv6. The proposed hosts in this document have an API or network-layer translator to allow existing IPv4 applications to communicate with IPv6-only peers. The BIH architecture consists of an Extension Name Resolver, an Address Mapper, and depending on implementation either a Function Mapper or a Protocol Translator. It is worth noting that the Extension Name Resolver's placement is orthogonal to the placement of protocol translation. For example, the Extension Name Resolver may reside in the socket API while protocol translation takes place at the networking layer.

2.1. Function Mapper

The function mapper translates an IPv4 socket API function into an IPv6 socket API function.

When detecting IPv4 socket API function calls from IPv4 applications, the function mapper MUST intercept the function calls and invoke IPv6 socket API functions that correspond to the IPv4 socket API functions.

The function mapper MUST NOT perform function mapping when the application is initiating communications to the address range used by local synthesis and the address mapping table does not have an entry

mathching the address.

See Appendix A for an informational list of functions that would be appropriate to intercept by the function mapper.

2.2. Protocol translator

The protocol translator translates IPv4 into IPv6 and vice versa using the IP conversion mechanism defined in SIIT [RFC6145]. To avoid unnecessary fragmentation, the host's IPv4 module should be configured with a small enough MTU (IPv6 link MTU - 20 bytes).

Protocol translation cannot be performed for IPv4 packets sent to the IPv4 address range used by local synthesis and for which a mapping table entry does not exist. The implementation SHOULD attempt to route such packets via IPv4 interfaces instead.

2.3. Extension Name Resolver

The Extension Name Resolver (ENR) returns an answer in response to the IPv4 application's name resolution request.

In the case of the socket API layer implementation option, when an IPv4 application tries to do a forward lookup to resolve names via the resolver library (e.g., gethostbyname()), BIH intercepts the function call and instead calls the IPv6 equivalent functions (e.g., getaddrinfo()) that will resolve both A and AAAA records. This implementation option is name resolution protocol agnostic, and hence supports techniques such as "hosts-file", NetBIOS, mDNS, and anything else the underlying operating system uses.

In the case of the network layer implementation option, the ENR intercepts the A query and creates an additional AAAA query with similar content. The ENR will then collect replies to both A and AAAA queries and, depending on results, either return an A reply unmodified or synthesize a new A reply. The network layer implementation option will only be able to catch applications' name resolution requests that result in actual DNS queries, hence is more limited when compared to the socket API layer implementation option. Hence the socket API layer option is RECOMMENDED.

In either implementation option, if there is a real IPv4 address available, the ENR SHOULD NOT synthesize IPv4 addresses. By default an ENR implementation MUST NOT synthesize IPv4 addresses when real IPv4 addresses exist.

If only IPv6 addresses are available for the queried name, the ENR asks the address mapper to assign a local IPv4 address corresponding to each IPv6 address.

In the case of the API layer implementation option, the ENR will simply make the API (e.g. gethostbyname) return the synthetic address. In the case of the network-layer implementation option, the ENR synthesizes an A record for the assigned IPv4 address, and delivers it up the stack. If the response contains a CNAME or a DNAME record, then the CNAME or DNAME chain is followed until the first terminating A or AAAA record is reached.

Application query	Network response	ENR behavior
IPv4 address(es)	IPv4 address(es)	return real IPv4 address(es)
IPv4 address(es)	IPv6 address(es)	synthesize IPv4 address(es)
IPv4 address(es)	IPv4/IPv6 address(es)	return real IPv4 address(es)

Figure 3: ENR behavior illustration

2.3.1. Special exclusion sets for A and AAAA records

An ENR implementation SHOULD by default exclude certain IPv4 and IPv6 addresses seen on received A and AAAA records. The addresses to be excluded by default MAY include addresses such as those that should not appear in the DNS or on the wire (see [RFC6147] section 5.1.4 and [RFC5735]). Additional addresses MAY be excluded based on possibly configurable local policies.

2.3.2. DNSSEC support

When the ENR is implemented at the network layer, the A record synthesis can cause similar issues as are described in [RFC6147] section 3. The main resolver of a host running BIH SHOULD NOT perform validation of A records, as it will be impossible to validate synthetic A records created by ENR. The ENR may support DNSSEC.

When the ENR is implemented at the socket API level, there are no issues with DNSSEC use, as the ENR itself uses socket APIs for DNS resolution. This approach is RECOMMENDED.

DNSSEC can also be supported by configuring the (stub) resolver on a host to trust validations done by the ENR located at network layer or alternatively the validating resolver can implement the ENR itself.

2.3.3. Reverse DNS lookup

When an application requests a reverse lookup for an IPv4 address, the ENR MUST check whether the queried IPv4 address can be found in

the Address Mapper's mapping table and is a local IPv4 address. If an entry is found and the queried address is locally generated, the ENR MUST initiate a corresponding reverse lookup for the real IPv6 address. In the case where the application requested a reverse lookup for an address not part of the local IPv4 address pool, e.g., a global address, the request MUST be passed on unmodified.

For example, when an application requests a reverse lookup for a synthesized locally valid IPv4 address, the ENR needs to intercept that query. The ENR asks the address mapper for the IPv6 address that corresponds to the IPv4 address. The ENR shall perform a reverse lookup procedure for the destination's IPv6 address and return the name received as a response to the application that initiated the IPv4 query.

2.3.4. DNS cache in network-layer ENR

When BIH shuts down, e.g., due to an IPv4 interface becoming available, BIH MUST flush the node's DNS cache of possible locally generated entries. This cache may be in the network-layer ENR itself, but also possibly host's caching stub resolver.

2.4. Address Mapper

The address mapper maintains a local IPv4 address pool. The pool consists of private IPv4 addresses per section 4.3. Also, the address mapper maintains a table consisting of pairs of locally selected IPv4 addresses and destinations' IPv6 addresses.

When the extension name resolver, translator, or the function mapper requests the address mapper to assign an IPv4 address corresponding to an IPv6 address, the address mapper selects and returns an IPv4 address out of the local pool, and registers a new entry into the table. The registration occurs in the following 3 cases:

(1) When the extension name resolver gets only IPv6 addresses for the target host name and there is no existing mapping entry for the IPv6 addresses. One or more local IPv4 addresses will be returned to the application and mappings for local IPv4 addresses to real IPv6

addresses are created.

(2) When the extension name resolver gets both IPv4 and IPv6 addresses, but the IPv4 addresses contain only excluded IPv4 addresses (e.g., 127.0.0.1). The behavior will follow case (1).

(3) When the function mapper gets a socket API function call triggered by a received IPv6 packet and there is no existing mapping entry for the IPv6 source address (for example, the client sent a UDP

request to an anycast address but a response was received from a unicast address).

Other possible combinations are outside of BIH and BIH is not involved in those.

3. Behavior and Network Examples

Figure 4 illustrates a very basic network scenario. An IPv4-only application is running on a host attached to the IPv6-only Internet and is talking to an IPv6-only server. Communication is made possible by Bump-In-the-Host.

```
 +----+                                     +-------------+
 | H1 |----------- IPv6 Internet -------- | IPv6 server |
 +----+                                     +-------------+
 v4 only
 application
```

Figure 4: Network Scenario #1

Figure 5 illustrates a possible network scenario where an IPv4-only application is running on a host attached to a dual-stack network, but the destination server is running on a private site that is numbered with public IPv6 addresses and private IPv4 addresses without port forwarding set up on the NAT44. The only means to contact the server is to use IPv6.

```
 +----------------------+  +------------------------------+
 | Dual Stack Internet  |  | IPv4 Private site (Net 10)   |
 |                      |  | IPv6 routed site             |
```

```
     |                   +---------+             +-----------+ |
     |                 +-|  NAT44  |-------------+           | |
     |  +----+         | +---------+             |           | |
     |  | H1 |---------+    |  |                 |  Server   | |
     |  +----+         | +-----------+           |           | |
     | v4 only         +-|IPv6 Router|-----------+           | |
     | application       +-----------+           +-----------+ |
     |                      |  |                  Dual Stack   |
     |                      |  |                    10.1.1.1   |
     |                      |  |                 2001:DB8::1   |
     +---------------------+  +-------------------------------+
```

Figure 5: Network Scenario #2

Illustrations of host behavior in both implementation options are given here. Figure 6 illustrates a setup where BIH (including the ENR) is implemented at the sockets API layer, and Figure 7 illustrates a setup where BIH (including the ENR) is implemented at the network layer.

```
 "dual stack"                                                "host6"
 IPv4    Socket |     [ API Translator ]    | TCP(UDP)/IP          Name
 appli-  API    | ENR      Address  Function| (v6/v4)             Server
 cation         |          Mapper   Mapper  |
  |        |        |        |        |         |              |       |
 <<Resolve IPv4 addresses for "host6".>>        |              |       |
  |        |        |        |        |         |              |       |
  |------->|------->|  Query IPv4 addresses for host6.         |       |
  |        |        |        |        |         |              |       |
  |        |        |------------------------------------------------->|
  |        |        |  Query 'A' and 'AAAA' records for host6          |
  |        |        |        |        |         |              |       |
  |        |        |<-------------------------------------------------|
  |        |        |  Reply with the 'AAAA' record.           |       |
  |        |        |        |        |         |              |
  |        |        |<<The 'AAAA' record is resolved.>>        |
  |        |        |        |        |         |              |
  |        |        |+++++++>|  Request one IPv4 address       |
```

```
  |        |        |        |  corresponding to the IPv6 address.
  |        |        |        |        |         |              |
  |        |        |        |<<Assign one IPv4 address.>>     |
  |        |        |        |        |         |              |
  |        |        |<+++++++|  Reply with the IPv4 address.   |
  |        |        |        |        |         |              |
  |<-------|<-------| Reply with the IPv4 address              |
  |        |        |        |        |         |              |
  |        |        |        |        |         |              |
 <<Call IPv4 Socket API function >>   |         |              |
  |        |        |        |        |         |              |
  |=======>|=========================>|An IPv4 Socket API function call
  |        |        |        |        |         |              |
  |        |        |        |<+++++++|  Request IPv6 addresses|
  |        |        |        |        |  corresponding to the  |
  |        |        |        |        |  IPv4 addresses.       |
  |        |        |        |        |         |              |
  |        |        |        |+++++++>| Reply with the IPv6 addresses.
  |        |        |        |        |         |              |
  |        |        |        |        |<<Translate IPv4 into IPv6.>>
  |        |        |        |        |         |              |
  |  An IPv6 Socket API function call.|=======================>|
  |        |        |        |        |         |              |
  |        |        |        |        |<<IPv6 data received    |
  |        |        |        |        |  from network.>>       |
  |        |        |        |        |         |              |
  |  An IPv6 Socket API function call.|<=======================|
  |        |        |        |        |         |              |
  |        |        |        |        |<<Translate IPv6 into IPv4.>>
  |        |        |        |        |         |              |
```

```
  |        |        |        |<+++++++|  Request IPv4 addresses|
  |        |        |        |        |  corresponding to the  |
  |        |        |        |        |  IPv6 addresses.       |
  |        |        |        |        |         |              |
  |        |        |        |+++++++>| Reply with the IPv4 addresses.
  |        |        |        |        |         |              |
  |<=======|<=========================|  An IPv4 Socket function call.
  |        |        |        |        |         |              |
```

Figure 6: Example of BIH as API addition

```
      "dual stack"                                         "host6"
   IPv4 stub  TCP/    ENR     address  translator  IPv6
   app  res.  IPv4            mapper
     |   |    |       |         |       |           |         |
   <<Resolve an IPv4 address for "host6".>>         |         |
     |-->|    |       |         |       |           |         |
     |   |----------->|  Query 'A' records for "host6".       |  Name
     |   |    |       |         |       |           |         |  Server
     |   |    |       |------------------------------------------->|
     |   |    |       |  Query 'A' and 'AAAA'  records for "host6"
     |   |    |       |         |       |           |         |    |
     |   |    |       |<-------------------------------------------|
     |   |    |       |  Reply only with 'AAAA' record.       |
     |   |    |       |         |       |           |         |
     |   |    |       |<<Only 'AAAA' record is resolved.>>    |
     |   |    |       |         |       |           |         |
     |   |    |       |-------->|  Request one IPv4 address   |
     |   |    |       |         |  corresponding to each IPv6 address.
     |   |    |       |         |       |           |         |
     |   |    |       |         |<<Assign IPv4 addresses.>>   |
     |   |    |       |         |       |           |         |
     |   |    |       |<--------|  Reply with the IPv4 address.
     |   |    |       |         |       |           |         |
     |   |    |       |<<Create 'A' record for the IPv4 address.>>
     |   |    |       |         |       |           |         |
     |   |<-----------|  Reply with the 'A' record. |         |
     |   |    |       |         |       |           |         |
     |<--|<<Reply with the IPv4 address |           |         |
     |   |    |       |         |       |           |         |
     <<Send an IPv4 packet to "host6".>>|           |         |
     |   |    |       |         |       |           |         |
     |=======>|========================>|  An IPv4 packet.    |
     |   |    |       |         |       |           |         |
     |   |    |       |         |<++++++|  Request IPv6 addresses
     |   |    |       |         |       |  corresponding to the IPv4
     |   |    |       |         |       |  addresses.         |
```

```
     |   |    |       |         |       |           |         |
     |   |    |       |         |++++++>|  Reply with the IPv6|
     |   |    |       |         |       |  addresses.         |
     |   |    |       |         |       |           |         |
```

```
     |   |    |       |         |       |<<Translate IPv4 into IPv6.>>
     |   |    |       |         |       |           |         |
     |   |    |       |An IPv6 packet.  |==========>|========>|
     |   |    |       |         |       |           |         |
     |   |    |       |         |   <<Reply with an IPv6 packet to.>>
     |   |    |       |         |       |           |         |
     |   |    |       |An IPv6 packet.  |<==========|<========|
     |   |    |       |         |       |           |         |
     |   |    |       |         |       |<<Translate IPv6 into IPv4.>>
     |   |    |       |         |       |           |         |
     |   |    |       |         |<++++++|Request IPv4 addresses
     |   |    |       |         |       |corresponding to the |
     |   |    |       |         |       |  IPv6 addresses.    |
     |   |    |       |         |       |           |         |
     |   |    |       |         |++++++>| Reply with the IPv4 addresses.
     |   |    |       |         |       |           |         |
     |<=======|=========================|  An IPv4 packet.    |
     |   |    |       |         |       |           |         |
```

Figure 7: Example of BIH at the network layer

4. Considerations

4.1. Socket API Conversion

IPv4 socket API functions are translated into IPv6 socket API functions that are semantically as identical as possible and vice versa. See Appendix B for the API list intercepted by BIH. However, some IPv4 socket API functions are not fully compatible with IPv6 since IPv4 supports features that are not present in IPv6, such as SO_BROADCAST.

4.2. Socket bindings

BIH SHOULD select a source address for a socket from the recommended source address pool if a socket used for communications has not been explicitly bound to any IPv4 address.

The binding of an explicitly bound socket MUST NOT be changed by the BIH.

4.3. ICMP Message Handling

When an application needs ICMP messages values (e.g., Type, Code, etc.) sent from the network layer, ICMPv4 message values MAY be translated into ICMPv6 message values based on SIIT [RFC6145], and vice versa.

4.4. IPv4 Address Pool and Mapping Table

The address pool consists of the private IPv4 addresses as per [RFC1918]. This pool can be implemented at different granularities in the node, e.g., a single pool per node, or at some finer granularity such as per-user or per-process. In the case of a large number of IPv4 applications communicating with a large number of IPv6 servers, the available address space may be exhausted if the granularity is not fine enough. This should be a rare event and chances will decrease as IPv6 support increases. The applications may use IPv4 addresses they learn for a much longer period than DNS time-to-live indicates. Therefore, the mapping table entries should be kept active for a long period of time. For example, a web browser may initiate one DNS query and then create multiple TCP sessions over time to the address it learns. When address mapping table clean-up is required the BIH may utilize techniques used by network address translators, such as described in [RFC2663], [RFC5382], and [RFC5508].

The RFC1918 address space was chosen because generally legacy applications understand it as a private address space. A new

dedicated address space would run a risk of not being understood by applications as private. 127/8 and 169.254/16 are rejected due to possible assumptions applications may make when seeing those.

The RFC1918 addresses used by the BIH have a risk of conflicting with addresses used in the host's possible IPv4 interfaces and corresponding local networks. The conflicts can be mitigated, but not fully avoided, by using less commonly used portions of the RFC1918 address space. Addresses from 172.16/12 are thought to be less likely to be in conflict than addresses from 10/8 or 192.168/16 spaces. A source address can usually be selected in a non-conflicting manner, but a small possibility exists for synthesized destination addresses being in conflict with real addresses used in local IPv4 networks.

The RECOMMENDED IPv4 addresses are following:

Primary source addresses: 172.21.112.0/20. Source addresses have to be allocated because applications use getsockname() calls and in the network layer mode an IP address of the IPv4 interface has to be shown (e.g., by 'ifconfig'). More than one address is allocated to allow implementation flexibility, e.g., for cases where a host has multiple IPv6 interfaces. The source addresses are from different subnets than destination addresses to ensure applications would not make on-link assumptions and would instead enable NAT traversal functions.

Secondary source addresses: 10.170.224.0/20. These addresses are recommended if a host has a conflict with primary source addresses.

Primary destination addresses: 10.170.160.0/20. The address mapper will select destination addresses primarily out of this pool.

Secondary destination addresses: 172.21.80.0/20. The address mapper will select destination addresses out of this pool if the node has a dual-stack connection conflicting with primary destination addresses.

4.5. Multi-interface

In the case of dual-stack destinations BIH MUST NOT do protocol translation from IPv4 to IPv6 when the host has any IPv4 interfaces, native or tunneled, available for use.

It is possible that an IPv4 interface is activated during BIH operation, for example if a node moves to a coverage area of an IPv4-enabled network. In such an event, BIH MUST stop initiating protocol translation sessions for new connections and BIH MAY disconnect active sessions. The choice of disconnection is left for

implementations and it may depend on whether IPv4 address conflict occurs between addresses used by BIH and addresses used by the new IPv4 interface.

4.6. Multicast

Protocol translation for multicast is not supported.

5. Considerations due ALG requirements

No ALG functionality is specified herein as ALG design is generally not encouraged for host-based translation and as BIH is intended for applications that do not include IP addresses in protocol payloads.

6. IANA Considerations

There are no actions for IANA.

---

7. Security Considerations

The security considerations of BIH follows closely, but not completely, those of NAT64 [RFC6146] and DNS64 [RFC6147]. The following sections are copied from RFC6146 and RFC6147 and modified for BIH scenario.

7.1. Implications on End-to-End Security

Any protocols that protect IP header information are essentially incompatible with BIH. This implies that end-to-end IPsec verification will fail when the Authentication Header (AH) is used (both transport and tunnel mode) and when ESP is used in transport mode. This is inherent in any network-layer translation mechanism. End-to-end IPsec protection can be restored, using UDP encapsulation as described in [RFC3948]. The actual extensions to support IPsec are out of the scope of this document.

7.2. Filtering

BIH creates binding state using packets flowing from the IPv4 side to the IPv6 side. In accordance with the procedures defined in this document following the guidelines defined in [RFC4787], a BIH implementation MUST offer "Endpoint-Independent Mapping".

Implementations MAY also provide support for "Address-Dependent Mapping" following the guidelines defined in [RFC4787].

The security properties, however, are determined by which packets the BIH allows in and which it does not. The security properties are determined by the filtering behavior and by the possible filtering configuration in the filtering portions of the BIH, not by the address mapping behavior.

7.3. Attacks on BIH

The BIH implementation itself is a potential victim of different types of attacks. In particular, the BIH can be a victim of DoS attacks. The BIH implementation has a limited number of resources that can be consumed by attackers creating a DoS attack. The BIH has a limited number of IPv4 addresses that it uses to create the bindings. Even though the BIH performs address and port translation, it is possible for an attacker to consume the synthetic IPv4 address pool by triggering a host to issue DNS queries for names that cause ENR to synthesise A records. DoS attacks can also affect other limited resources available in the host running BIH such as memory or link capacity. For instance, it is possible for an attacker to launch a DoS attack on the memory of the BIH running device by

sending fragments that the BIH will store for a given period. If the number of fragments is high enough, the memory of the host could be exhausted. BIN implementations MUST implement proper protection against such attacks, for instance, allocating a limited amount of memory for fragmented packet storage.

Another consideration related to BIH resource depletion refers to the preservation of binding state. Attackers may try to keep a binding state alive forever by sending periodic packets that refresh the state. In order to allow the BIH to defend against such attacks, the BIH implementation MAY choose not to extend the session entry lifetime for a specific entry upon the reception of packets for that entry through the external interface.

7.4. DNS considerations

BIH operates in combination with the DNS, and is therefore subject to whatever security considerations are appropriate to the DNS mode in which the BIH is operating (i.e., authoritative, recursive, or stub-resolver mode).

BIH has the potential to interfere with the functioning of DNSSEC, because BIH modifies DNS answers, and DNSSEC is designed to detect such modifications and to treat modified answers as bogus.

8. Changes since RFC2767 and RFC3338

This document combines and obsoletes both [RFC2767] and [RFC3338].

The changes in this document mainly reflect the following components:

1. Supporting IPv6-only network connections

2. The IPv4 address pool uses private address instead of reserved IPv4 addresses (0.0.0.1 - 0.0.0.255)

3. Extending ENR and address mapper to operate differently

4. Adding an alternative way to implement the ENR

5. Standards track instead of experimental/informational

6. Supporting reverse (PTR) queries

9. Acknowledgments

The authors thank the discussion from Gang Chen, Dapeng Liu, Bo Zhou, Hong Liu, Tao Sun, Zhen Cao, Feng Cao et al. in the development of this document.

The efforts of Mohamed Boucadair, Dean Cheng, Lorenzo Colitti, Paco Cortes, Marnix Goossens, Ala Hamarsheh, Ed Jankiewizh, Suresh Krishnan, Julien Laganier, Yiu L. Lee, Jan M. Melen, Qibo Niu, Pierrick Seite, Christian Vogt, Magnus Westerlund, Dan Wing, Dave Harrington, and James Woodyatt in reviewing this document are gratefully acknowledged.

Special acknowledgements go to Dave Thaler for his extensive review and support.

The authors of RFC2767 acknowledged WIDE Project, Kazuhiko YAMAMOTO, Jun MURAI, Munechika SUMIKAWA, Ken WATANABE, and Takahisa MIYAMOTO. The authors of RFC3338 acknowledged implementation contributions by Wanjik Lee (wjlee@arang.miryang.ac.kr) and i2soft Corporation (www.i2soft.net).

The authors of Bump-in-the-Wire (BIW) (draft-ietf-biw-00.txt, October 2006), P. Moster, L. Chin, and D. Green, are acknowledged. Some ideas and clarifications from BIW have been adapted to this document.

10. References

10.1. Normative References

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC4213] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", RFC 6145, April 2011.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, April 2011.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, April 2011.

10.2. Informative References

[RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.

[RFC2767] Tsuchiya, K., HIGUCHI, H., and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)", RFC 2767, February 2000.

[RFC3338] Lee, S., Shin, M-K., Kim, Y-J., Nordmark, E., and A. Durand, "Dual Stack Hosts Using "Bump-in-the-API" (BIA)", RFC 3338, October 2002.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W.

Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[RFC5382] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, October 2008.

[RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, April 2009.

[RFC5735] Cotton, M. and L. Vegoda, "Special Use IPv4 Addresses", BCP 153, RFC 5735, January 2010.

[RFC6180] Arkko, J. and F. Baker, "Guidelines for Using IPv6 Transition Mechanisms during IPv6 Deployment", RFC 6180, May 2011.

Appendix A. API list intercepted by BIH

The following informational list includes API functions that would be appropriate to intercept by BIH module when implemented at socket layer. Please note that this list may not be fully exhaustive.

The functions that the application uses to pass addresses into the system are:

`bind()`

`connect()`

`sendmsg()`

`sendto()`

`gethostbyaddr()`

`getnameinfo()`

The functions that return an address from the system to an application are:

accept()

recvfrom()

recvmsg()

getpeername()

getsockname()

gethostbyname()

getaddrinfo()

The functions that are related to socket options are:

getsocketopt()

setsocketopt()

As well, raw sockets for IPv4 and IPv6 MAY be intercepted.

Most of the socket functions require a pointer to the socket address structure as an argument. Each IPv4 argument is mapped into corresponding an IPv6 argument, and vice versa.

According to [RFC3493], the following new IPv6 basic APIs and structures are required.

IPv4	new IPv6
AF_INET	AF_INET6
sockaddr_in	sockaddr_in6
gethostbyname()	getaddrinfo()
gethostbyaddr()	getnameinfo()
inet_ntoa()/inet_addr()	inet_pton()/inet_ntop()
INADDR_ANY	in6addr_any

Figure 8

BIH MAY intercept inet_ntoa() and inet_addr() and use the address mapper for those. Doing that enables BIH to support literal IP addresses. However, IPv4 address literals can only be used after a mapping entry between the IPv4 address and corresponding IPv6 address has been created.

The gethostbyname() and getaddrinfo() calls return a list of addresses. When the name resolver function invokes getaddrinfo() and getaddrinfo() returns multiple IP addresses, whether IPv4 or IPv6, they SHOULD all be represented in the addresses returned by gethostbyname(). Thus if getaddrinfo() returns multiple IPv6 addresses, this implies that multiple address mappings will be created; one for each IPv6 address.

Authors' Addresses

Bill Huang
China Mobile
53A,Xibianmennei Ave.,
Xuanwu District,
Beijing 100053

China

Email: bill.huang@chinamobile.com

Hui Deng
China Mobile
53A,Xibianmennei Ave.,
Xuanwu District,
Beijing 100053
China

Email: denghui02@gmail.com

Teemu Savolainen
Nokia
Hermiankatu 12 D
FI-33720 TAMPERE
Finland

Email: teemu.savolainen@nokia.com