Basic Socket Interface Extensions for Host Identity Protocol (HIP)

draft-ietf-hip-native-api-06

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 23, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document defines extensions to the current sockets API for Host Identity Protocol (HIP). The extensions focus on the use of public-key based identifiers discovered via DNS resolution, but define also interfaces for manual bindings between HITs and locators. With the extensions, the application can also support more relaxed security models where the communication can be non-HIP based, according to local policies. The extensions in document are experimental and provide basic tools for futher experimentation with policies.

Table of Contents

1. Introduction 3

2. Terminology 4

3. API Overview 4
   3.1. Interaction with the Resolver 5
   3.2. Interaction without a Resolver 5

4. API Syntax and Semantics 6
   4.1. Socket Family and Address Structure Extensions 6
   4.2. Extensions to Resolver Data Structures 8
      4.2.1. Resolver Usage 9
   4.3. The Use of getsockname and getpeername Functions 10
   4.4. Validating HITs 10
   4.5. Source HIT Selection by the System 11
   4.6. Explicit Handling of Locators 12

5. Summary of New Definitions 14

6. IANA Considerations 15

7. Security Considerations 15

8. Contributors 15

9. Acknowledgements 15

10. Normative References 16

Authors' Addresses 17

1. Introduction

This document defines C-based sockets Application Programming Interface (API) extensions for handling HIP-based identifiers explicitly in HIP-aware applications. It is up to the applications, or high-level programming languages or libraries, to manage the identifiers. The extensions in this document are mainly related to the use case in which a DNS resolution step has occurred prior to the creation of a new socket, and assumes that the system has cached or is otherwise able to resolve identifiers to locators (IP addresses). The DNS extensions for HIP are described in [RFC5205]. The extensions also cover the case in which an application may want to explicitly provide suggested locators with the identifiers, including supporting the opportunistic case in which the system does not know the peer host identity.

The Host Identity Protocol (HIP) [RFC4423] proposes a new cryptographic namespace by separating the roles of end-point identifiers and locators by introducing a new namespace to the TCP/IP stack. SHIM6 [I-D.ietf-shim6-proto] is another protocol based on identity-locator split. Note that the APIs specified in this document are specific to HIP. However, the APIs here have been designed as much as possible so as not to preclude its use with other protocols. The use of these APIs with other protocols is, nevertheless, for further study.

Applications can observe the HIP layer and its identifiers in the networking stacks with varying degrees of visibility. [RFC5338] discusses the lowest levels of visibility in which applications are completely unaware of the underlying HIP layer. Such HIP-unaware applications in some circumstances use HIP-based identifiers, such as LSIs or HITs, instead of IPv4 or IPv6 addresses and cannot observe the identifier-locator bindings.

This document specifies extensions to [RFC3493] to define a new socket address family, AF_HIP. The macro AF_HIP is used as an alias

for PF_HIP in this document because the distinction between AF and PF has been lost in practice. The extensions also describe a new socket address structure for sockets using Host Identity Tags (HITs) explicitly and describe how the socket calls in [RFC3493] are adapted or extended as a result.

Some applications may accept incoming communications from any identifier. Other applications may initiate outgoing communications without the knowledge of the peer identifier in Opportunistic Mode [RFC5201] by just relying on a peer locator. This document describes how to address both situations using "wildcards" as described later in this document.

There are two related API documents. Multihoming and explicit locator-handling related APIs are defined in [I-D.ietf-shim6-multihome-shim-api]. IPsec related policy attributes and channel bindings APIs are defined in [I-D.ietf-btns-c-api]. Most of the extensions defined in this document can be used independently of the two mentioned related API documents.

The identity-locator split introduced by HIP introduces some policy related challenges with datagram oriented sockets, opportunistic mode, and manual bindings between HITs and locators. The extensions in this document are of experimental nature and provide basic tools for experimenting with policies. Policy related issues are left for further experimentation.

To recap, the extensions in this document have three goals. The first goal is to allow HIP-aware applications to open sockets to other hosts based on the HITs alone, presuming that the underlying system can resolve the HITs to addresses used for initial contact. The second goal is that applications can explicitly initiate communications with unknown peer identifiers. The third goal is to define how HIP-aware applications may provide suggested initial contact addresses along with the HITs.

2. Terminology

The terms used in this document are summarized in Table 1.

```
+---------+--------------------------------------------------------+
```

Term	Explanation
HIP	Host Identity Protocol
HIT	Host Identity Tag, a 100-bit hash of a public key with a 28 bit prefix
LSI	Local Scope Identifier, a local, 32-bit descriptor for a given public key.
Locator	Routable IPv4 or IPv6 address used at the lower layers

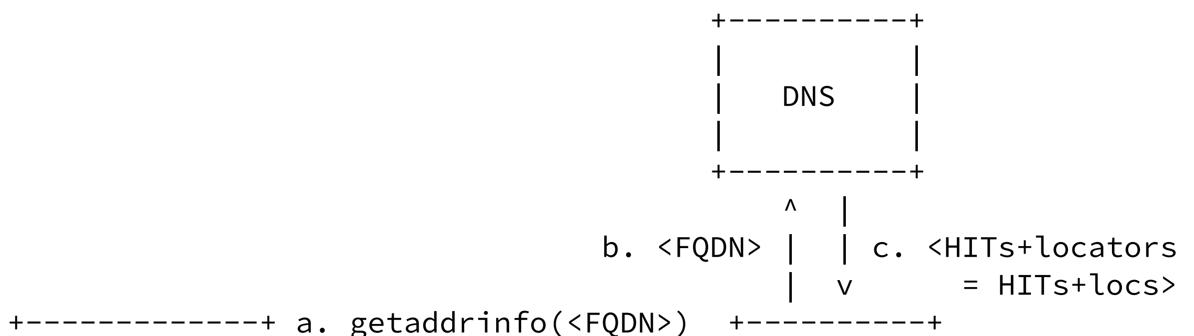
Table 1

3. API Overview

This section provides an overview of how the API can be used. First, the case in which a resolver is involved in name resolution is described, and then the case in which no resolver is involved is described.

3.1. Interaction with the Resolver

Before an application can establish network communications with the entity named by a given FQDN or relative host name, the application must translate the name into the corresponding identifier(s). DNS-based hostname-to-identifier translation is illustrated in Figure 1. The application calls the resolver in step a to resolve an FQDN step b. The DNS server responds with a list of HITs and a set of locators step c. Optionally in step d, the resolver caches the HIT to locator mapping to the HIP module. The resolver returns the HITs to the application step e. Finally, the application selects one HIT and uses it in a socket call such as connect() in step f.



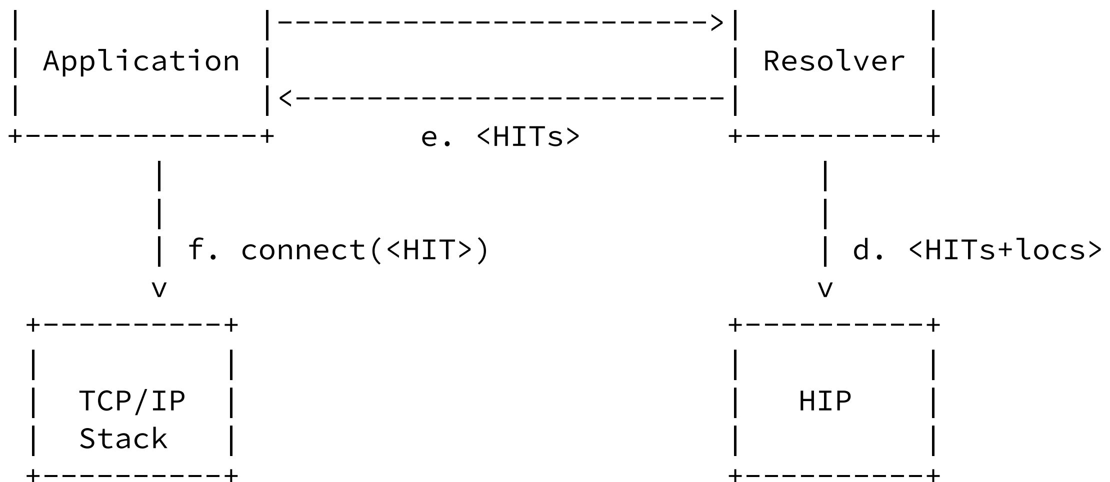


Figure 1

In practice, the resolver functionality can be implemented in different ways. For example, it may be implemented in existing resolver libraries or as a DNS proxy.

3.2. Interaction without a Resolver

The extensions in this document focus on the use of the resolver to map host names to HITs and locators in HIP-aware applications. The resolver associates implicitly the HIT with the locator(s) by e.g. communicating the HIT-to-IP mapping to the HIP daemon. However, it

is possible that an application operates directly on a peer HIT without interacting with the resolver. In such a case, the application may resort to the system to map the peer HIT to an IP address. Alternatively, the application can explicitly map the HIT to an IP address using socket options as specified in [I-D.ietf-shim6-multihome-shim-api]. Full support for all of the extensions defined in this draft requires shim socket options to be implemented by the system.

4. API Syntax and Semantics

In this section, we describe the native HIP APIs using the syntax of the C programming language. We limit the description to the interfaces and data structures that are either modified or completely

new, because the native HIP APIs are otherwise identical to the sockets API [POSIX].

4.1. Socket Family and Address Structure Extensions

The sockets API extensions define a new protocol family, PF_HIP, and a new address family, AF_HIP. The AF_HIP and PF_HIP are aliases to each other. These definition shall be defined as a result of including <sys/socket.h>.

The use of the PF_HIP constant is mandatory with the socket() function when an application uses the native HIP APIs. The application gives the PF_HIP constant as the first argument (domain) to the socket() function. The system returns a positive integer representing a socket descriptor when the system supports HIP. Otherwise, the system returns -1 and sets errno to EAFNOSUPPORT.

Figure 2 shows socket address structure for HIP.

```
#include <netinet/in.h>

typedef struct in6_addr hip_hit_t;

struct sockaddr_hip {
          sa_family_t    ship_family;
          in_port_t      ship_port;
          uint32_t       ship_pad;
          uint64_t       ship_flags;
          hip_hit_t      ship_hit;
          uint8_t        ship_reserved[16];
};
```

Figure 2

Figure 2 is in in 4.3BSD format. The family of the socket, ship_family, is set to AF_HIP. The port number ship_port is two octets in network byte order. and the ship_hit is 16 octets in network byte order. An implementation may have extra member(s) in this structure.

The application usually sets the ship_hit field using the resolver. However, the application can use three special wildcard macros to set

a value directly into the ship_hit field. The macros are HIP_HIT_ANY, HIP_HIT_ANY_PUB, HIP_HIT_ANY_TMP and HIP_ADDR_ANY. The first three equal to a HIT value associated with a wildcard HIT of any, public, or anonymous type. The fourth macro, HIP_ADDR_ANY, denotes both HIP_HIT_ANY or any IPv4 or IPv6 address. The HIP_HIT_ANY equals to HIP_HIT_ANY_PUB or HIP_HIT_ANY_TMP. The anonymous identifiers refer to the use anonymous identifiers as specified in [RFC4423]. The system may designate anonymous identifiers as meta data associated with a HIT depending on whether it has been published or not. However, there is no difference in the classes of HITs from the HIP protocol perspective,

The application can use the HIP_HIT_ANY_* and HIP_ADDR_ANY macros to accept incoming communications to all of the HITs of the local host. Incoming communications refers here to the functions such as bind(), recvfrom() and recvmsg(). The HIP_HIT_* macros are similar to the sockets API macros INADDR_ANY and IN6ADDR_ANY_INIT, but they are applicable to HITs only. After initial contact with the peer, the application can discover the local and peer HITs using getsockname() and getpeername() calls in the context of connection oriented sockets. The difference between the use of the HIP_HIT_* and HIP_ADDR_ANY macros here is that the former allows only HIP-based communications but the latter also allows communications without HIP.

The application also uses the HIP_HIT_ANY macro in ship_hit field to establish outgoing communications in Opportunistic mode [RFC5201], i.e., when the application knows the remote peer locator but not the HIT. Outgoing communications refers here to the use of functions such as connect(), sendto() and sendmsg(). However, the application should first associate the socket with at least one IP address of the peer using SHIM_LOCLIST_PEER_PREF socket option. The use of the HIP_HIT_ANY macro guarantees that the communications will be based on HIP or none at all.

The use of HIP_ADDR_ANY macro in the context of outgoing communications is left for further experimentation. It could be used for establishing a non-HIP based connectivity when HIP-based connectivity was unsuccessful.

Some applications rely on system level access control, either

implicit or explicit (such as accept_filter() function found on BSD-

based systems), but such discussion is out of scope. Other applications implement access control themselves by using the HITs. In such a case, the application can compare two HITs using memcmp() or similar function. It should be noticed that different connection attempts between the same two hosts can result in different HITs because a host is allowed to have multiple HITs.

4.2. Extensions to Resolver Data Structures

The HIP APIs introduce a new addrinfo flag, HIP_PREFER_ORCHID, to be used by application to query for both HIT and locator information via the getaddrinfo() resolver function [RFC3493]. The getaddrinfo() function uses a data structure used for both input to and output from the resolver. The data structure is illustrated in Figure 3.

```
#include <netdb.h>

struct addrinfo {
    int       ai_flags;           /* e.g. AI_EXTFLAGS */
    int       ai_family;          /* e.g. AF_HIP */
    int       ai_socktype;        /* e.g. SOCK_STREAM */
    int       ai_protocol;        /* 0 or IPPROTO_HIP */
    socklen_t ai_addrlen;         /* size of *ai_addr  */
    struct    sockaddr *ai_addr;  /* sockaddr_hip */
    char     *ai_canonname;      /* canon. name of the host */
    struct    addrinfo *ai_next;  /* next endpoint */
    int       ai_eflags;          /* RFC5014 extension */
};
```

Figure 3

Application must set both the flag AI_EXTFLAGS [RFC5014] in ai_flags and HIP_PREFER_ORCHID in the ai_eflags, or otherwise the resolver does not return sockaddr_hip data structures. The resolver returns EAI_BADFLAGS when it does not support HIP_PREFER_ORCHID or AI_EXTFLAGS flags.

The system may have a HIP-aware interposing DNS agent as described in section 3.2 in [RFC5014]. In such a case, the DNS agent returns transparently LSIs or HITs in sockaddr_in and sockaddr_in6 structures when available. To disable this behaviour, the application sets AI_EXTFLAGS and AI_NO_ORCHID flags.

Application denotes its preference for public and anonymous types of HITs using HIP_PREFER_SRC_PUBLIC and HIP_PREFER_SRC_TMP flags in the ai_eflags field. If the application sets neither of the flags, the resolver returns both public and anonymous HITs.

The simultaneous use of both HIP_PREFER_ORCHID and HIP_PREFER_PASSIVE_* flags produces a single sockaddr_hip structure containing a wildcard address that the application can use either for incoming (node argument is NULL in getaddrinfo) or outgoing communications (node argument is non-NULL). For example, HIP_PREFER_PASSIVE_HIT_TMP flag produces one sockaddr_hip structure that contains a HIP_HIT_ANY_TMP in the ship_hit field.

The resolver sets the ai_family field to AF_HIP in the addrinfo structure when ai_addr points to a sockaddr_hip structure.

When ai_protocol field is set to zero, the resolver also returns locators in sockaddr_in and sockaddr_in6 structures in addition to sockaddr_hip structures. The resolver returns only sockaddr_hip structures when the application has set the ai_protocol field to IPPROTO_HIP or a sockaddr_hip structure is given as the hint argument to the resolver.

4.2.1. Resolver Usage

A HIP-aware application creates the sockaddr_hip structures explicitly or obtains them from the resolver. The explicit configuration of locators is described in [I-D.ietf-shim6-multihome-shim-api]. This document defines "automated" resolver extensions for getaddrinfo() resolver [RFC3493].

```
#include <netdb.h>

int getaddrinfo(const char *nodename,
                const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res)
void free_addrinfo(struct addrinfo *res)
```

Figure 4

As described in [RFC3493], the getaddrinfo function takes the nodename, servname, and hints as its input arguments. It places the result of the query into the res argument. The return value is zero on success, or a non-zero error value on error. The nodename argument specifies the host name to be resolved; a NULL argument denotes the local host. The servname parameter declares the port number to be set in the socket addresses in the res output argument. Both the nodename and servname cannot be NULL.

The input argument "hints" acts like a filter that defines the

attributes required from the resolved endpoints. A NULL hints argument indicates that any kind of endpoints are acceptable.

The output argument "res" is dynamically allocated by the resolver. The application frees res argument with the free_addrinfo function. The res argument contains a linked list of the resolved endpoints. The linked list contains sockaddr_hip structures only when the input argument has the HIP_PREFER_ORCHID flag set in ai_eflags. The resolver inserts HITs before any locators. When the HIP_PREFER_ORCHID flag is set, the resolver does not return LSIs or HITs encapsulated into sockaddr_in or sockaddr_in6 data structures as described in [RFC5338].

Resolver can return a HIT which maps to multiple locators. The resolver may cache the locator mappings to the HIP module. The HIP module manages the multiple locators according to system policies of the host. The multihoming document [I-D.ietf-shim6-multihome-shim-api] describes how an application can override system default policies.

It should be noticed that the application can configure the HIT explicitly without setting the locator or the resolver can fail to resolve any locator. In this scenario, the application relies on the system to map the HIT to an IP address. When the system fails to provide the mapping, it returns -1 in the called sockets API function to the application and sets errno to EADDRNOTAVAIL.

4.3. The Use of getsockname and getpeername Functions

The application usually discovers the local or peer HITs from the sockaddr_hip structures returned by getaddrinfo(). However, the sockaddr_hip structure does not contain a HIT when the application uses the HIP_HIT_ANY_* macros. In such a case, the application discovers the local and peer HITs using the getsockname() and getpeername() functions. The functions return sockaddr_hip structures when the family of the socket is AF_HIP.

4.4. Validating HITs

An application that uses the HIP_ADDR_ANY macro may want to check if the local or peer address is an orchid-based HIT [RFC4843]. Also, the application may want to verify whether a HIT is public or

anonymous. The application accomplishes these using a new function called sockaddr_is_srcaddr() which is illustrated in Figure 5.

```
#include <netinet/in.h>

short sockaddr_is_srcaddr(struct sockaddr *srcaddr
                          uint64_t flags);
```

Figure 5

The sockaddr_is_srcaddr() function operates in the same way as inet6_is_srcaddr() function [RFC5014] which can be used to verify the type of an address belonging to the localhost. The difference is that sockaddr_is_srcaddr() function handles sockaddr_hip structures in addition to sockaddr_in6, and possibly some other socket structures in further extensions. The function has also 64 bit flags instead of 32 bits. This new function handles the same flags as defined in [RFC5014] in addition to some HIP-specific flags listed in Table 2.

Flag	Purpose
HIP_PREFER_ORCHID	The identifier is a HIT
HIP_PREFER_SRC_TMP	Anonymous HIT
HIP_PREFER_SRC_PUBLIC	Public HIT

Table 2

4.5. Source HIT Selection by the System

Some applications initiate communications by specifying only the destination identifier and let the underlying system specify the source. When the system selects the source HIT, the system should apply the rules specified in [RFC3484] according to the default policy table for HITs shown in Table 3.

HIT Type	Precedence	Label
Anonymous DSA	110	5

Anonymous RSA	120	6
Public DSA	130	7
Public RSA	140	8
[RFC3484] rules	50-100	7

Table 3

When application using a AF_HIP-based socket does not specify the source identifier, the system selects the source identifier on the behalf of the application according to the precedence in the above table. For example, the system prefers public (published) keys before anonymous keys because they work better for referral purposes. RSA-based keys are preferred over DSA based because RSA is the default algorithm in HIP.

When system provides multiple keys of same type, but with different key lengths, the longer keys should have a higher preference. As example, system providing two public RSA keys of different size would give the smaller key preference value 140 and 145 for the larger. The preference value should not exceed 150. Systems supporting more than 10 keys of same key size may use digits to further fragment the precedence namespace. IPv6 addresses have the lowest precedence value to denote that HITs have a higher precedence when operating on AF_HIP-based sockets.

[RFC5014] specifies flags for the getaddrinfo resolver and socket options for Mobile IPv6. The resolver, operating under HIP_PREFER_ORCHID flag, or the socket handler, operating on a AF_HIP-based socket, may encounter such flags or options. In such a case the resolver or socket handler should silenty ignore the flags or options without returning an error. However, a HIP-aware application may use the HIP-specific flags HIP_PREFER_ORCHID, HIP_PREFER_SRC_TMP or HIP_PREFER_SRC_PUBLIC in getsockopt(), setsockopt(), getaddrinfo() calls and in the anchillary data of datagram packets as specified in [RFC5014]. The level of the socket options should be set to SOL_SHIM [I-D.ietf-shim6-multihome-shim-api] and the option name should be HIP_HIT_PREFERENCES.

4.6. Explicit Handling of Locators

The system resolver, or the HIP module, maps HITs to locators implicitly. However, some applications may want to specify initial locator mappings explicitly. In such a case, the application first creates a socket with AF_HIP as the domain argument. Second, the application may set locator information with one of the following shim socket options as defined in the multihoming extensions in [I-D.ietf-shim6-multihome-shim-api]:

optname	get	set	description	dtype
SHIM_LOC_LOCAL_PREF	o	o	Get or set the preferred locator on the local side for the context associated with the socket.	*1
SHIM_LOC_PEER_PREF	o	o	Get or set the preferred locator on the remote side for the context associated with the socket.	*1
SHIM_LOCLIST_LOCAL	o	o	Get or set a	*2

			list of locators associated with the local EID.	
SHIM_LOCLIST_PEER	o	o	Get or set a list of locators associated with the peer's EID.	*2
SHIM_LOC_LOCAL_SEND	o	o	Request use of specific locator as source locator of outgoing IP packets.	*2
SHIM_LOC_PEER_SEND	o	o	Request use of specific locator as destination locator of outgoing IP packets.	*2

*1: Pointer to a shim_locator which is defined in Section 7 of draft-ietf-shim6-multihome-shim-api.
*2: Pointer to an array of shim_locator.

Figure 6

Finally, the application creates a valid sockaddr_hip structure and

associates the socket also with the sockaddr_hip structure by calling some socket-related function, such as connect() or bind().

The usage and semantics for typical use cases are as follows:

An application that initiates a connection using a connection oriented socket to a particular host at a known address or set of addresses can invoke SHIM_LOCLIST_PEER socket option. The HIP module uses the first address (if multiple are provided, or else the application can override this by setting SHIM_LOC_PEER_PREF to one of the addresses in SHIM_LOCLIST_PEER. The application later provides a

specific HIT in the ship_hit field of the sockaddr_hip in the connect() system call. If the application provides one or more addresses in SHIM_LOCLIST_PEER setsockopt call, the system should not connect to the host via another destination address, in case the application intends to restrict the range of addresses permissible as a policy choice. If the system cannot reach the provided HIT at one of the addresses provided, the outbound socket API functions (connect, sendmsg, etc.) return -1 and set errno to EINVALIDLOCATOR.

Another common use case is to set up an association in opportunistic mode, when the destination HIT is specified as a wildcard. This can be accomplished by setting one or more destination addresses using the SHIM_LOCLIST_PEER socket option as described above and then calling connect() with the wildcard HIT. The connect() call returns -1 and sets errno to EADDRNOTAVAIL when the application connects to a wildcard without specifying any destination address.

Applications may also choose to associate local addresses with sockets. The procedures specified in [I-D.ietf-shim6-multihome-shim-api] are followed in this case.

5. Summary of New Definitions

Table 4 summarizes the new macro and structures defined in this document.

Header	Definition
<sys/socket.h>	AF_HIP

<sys/socket.h>	PF_HIP
<netinet/in.h>	IPPROTO_HIP
<netinet/hip.h>	HIP_HIT_ANY
<netinet/hip.h>	HIP_HIT_ANY_PUB
<netinet/hip.h>	HIP_HIT_ANY_TMP
<netinet/hip.h>	HIP_ADDR_ANY
<netinet/hip.h>	HIP_HIT_PREFERENCES
<netinet/hip.h>	hip_hit_t
<netdb.h>	HIP_PREFER_ORCHID
<netdb.h>	HIP_PREFER_SRC_TMP
<netdb.h>	HIP_PREFER_SRC_PUBLIC
<netdb.h>	HIP_PREFER_PASSIVE_HIT_TMP
<netdb.h>	HIP_PREFER_PASSIVE_HIT_PUB
<netdb.h>	HIP_PREFER_PASSIVE_HIT_ANY
<netdb.h>	HIP_PREFER_PASSIVE_ADDR_ANY
<netinet/hip.h>	sockaddr_hip
<netinet/hip.h>	sockaddr_is_srcaddr

Table 4

6. IANA Considerations

No IANA considerations.

7. Security Considerations

No security considerations currently.

8. Contributors

Thanks for Jukka Ylitalo and Pekka Nikander for their original contribution, time and effort to the native HIP APIs. Thanks for Yoshifuji Hideaki for his contributions to this document.

9. Acknowledgements

Kristian Slavov, Julien Laganier, Jaakko Kangasharju, Mika Kousa, Jan Melen, Andrew McGregor, Sasu Tarkoma, Lars Eggert, Joe Touch, Antti Jaervinen, Anthony Joseph, Teemu Koponen, Jari Arkko, Ari Keraenen,

Juha-Matti Tapio, Shinta Sugimoto, Philip Matthews, Jan Melen and
Gonzalo Camarillo have also provided valuable ideas or feedback.
Thanks also for the APPS area folks, including Stephane Bortzmeyer,
Chris Newman, Tony Finch, "der Mouse" and Keith Moore.

10. Normative References

[I-D.ietf-btns-c-api]
Richardson, M., Williams, N., Komu, M., and S. Tarkoma, "C-Bindings for IPsec Application Programming Interfaces", draft-ietf-btns-c-api-04 (work in progress), March 2009.

[I-D.ietf-shim6-multihome-shim-api]
Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, "Socket Application Program Interface (API) for Multihoming Shim", draft-ietf-shim6-multihome-shim-api-08 (work in progress), May 2009.

[I-D.ietf-shim6-proto]
Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", draft-ietf-shim6-proto-12 (work in progress), February 2009.

[POSIX] Institute of Electrical and Electronics Engineers, "IEEE Std. 1003.1-2001 Standard for Information Technology - Portable Operating System Interface (POSIX)", Dec 2001.

[RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC4423] Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture", RFC 4423, May 2006.

[RFC4843] Nikander, P., Laganier, J., and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)", RFC 4843, April 2007.

[RFC5014] Nordmark, E., Chakrabarti, S., and J. Laganier, "IPv6 Socket API for Source Address Selection", RFC 5014, September 2007.

[RFC5201] Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson,

"Host Identity Protocol", RFC 5201, April 2008.

[RFC5205] Nikander, P. and J. Laganier, "Host Identity Protocol (HIP) Domain Name System (DNS) Extensions", RFC 5205, April 2008.

[RFC5338] Henderson, T., Nikander, P., and M. Komu, "Using the Host Identity Protocol with Legacy Applications", RFC 5338, September 2008.

Authors' Addresses

Miika Komu
Helsinki Institute for Information Technology
Metsaenneidonkuja 4
Helsinki
Finland

Phone: +358503841531
Fax: +35896949768
Email: miika@iki.fi
URI: http://www.iki.fi/miika/

Thomas Henderson
The Boeing Company
P.O. Box 3707
Seattle, WA
USA

Email: thomas.r.henderson@boeing.com