Host Identity Protocol
Internet-Draft
Intended status: Informational
Expires: January 8, 2008

M. Komu
Helsinki Institute for Information Technology

July 7, 2007

Native Application Programming Interfaces for SHIM APIs
draft-ietf-hip-native-api-02

Status of this Memo

Copyright Notice

Abstract

This document defines extensions to the current networking APIs for protocols based on identifier/locator split. Currently, the document focuses on HIP, but the extensions can be used also by other protocols implementing identifier locator split. Using the API extensions, new SHIM aware applications can configure manually

mappings between upper layer identifiers and the corresponding locators. Also, the API describes how to handle outbound connection establishment where an application is unaware of the peer identifier but knows the peer locator.

Table of Contents

1. Terminology 3

2. Introduction 3

3. Design Model 4
3.1. Layering Model 4
3.2. Namespace Model 4
3.3. Interaction with the Resolver 5

4. API Syntax and Semantics 6
4.1. Socket Family and Address Structure 6
4.2. Resolver 8

5. IANA Considerations 10

6. Security Considerations 10

7. Acknowledgements 10

8. Normative References 10

Author's Address 11
Intellectual Property and Copyright Statements 12

1. Terminology

Term	Explanation
ULID	Upper Layer IDentifier equals to the identity part of the identity/locator split. It is the identifier used by the application to name a peer for the transport layer.
Locator	Non-routable IPv4 or IPv6 address used at the lower layers
FQDN	Fully Qualified Domain Name
HIT	Host Identity Tag, a 100-bit hash of a public key with a 28 bit prefix
LSI	Locally Scope Identifier, a 32-bit local presentation of a public key

Table 1

2. Introduction

Host Identity Protocol proposes a new cryptographic namespace and a new layer to the TCP/IP architecture. Applications can see these new changes in the networking stacks with varying degrees of visibility. [I-D.henderson-hip-applications] discusses the lowest levels of visibility in which applications are either unaware of HIP. The HIP-unaware applications use LSIs or HITs instead of IPv4 or IPv6 addresses. Such applications may be unaware of the locator bindings.

This document discusses about visibility of HIP to HIP-aware applications. The applications are completely HIP aware and can control the HIP layer and Host Identifiers. This document defines C-based sockets API extensions for handling the bindings explicitly. The extensions expose the identity-locator split to SHIM-aware

applications and it is up to the application, or a higher level programming language, to manage the identities and locators.

The API extensions introduce a new socket address structure. The structure requires a new address family, PF_SHIM, for sockets that use HITs and locators explicitly. An application can also use the family to detect SHIM support in the local host.

Hosts may accept incoming or initiate outgoing communications without the knowledge of the identity of the peer. This document describes also how to address this situation using late binding based on new wildcards.

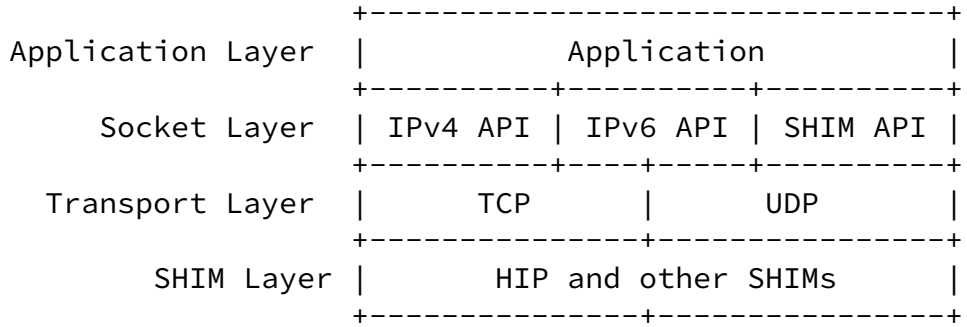
There are two related IETF documents that are define other related API extensions. Multihoming related APIs are defined in [I-D.ietf-shim6-multihome-shim-api]. IPsec related policy attributes and channel bindings are defined in [I-D.ietf-btns-c-api]

3. Design Model

In this section, the native SHIM API design is described from an design point of view. We describe the layering and namespace model. We conclude the discussion with a description of the resolver model.

3.1. Layering Model

The application layer accesses the transport layer via the socket interface. The application layer uses the traditional TCP/IP IPv4 or IPv6 interface, or the new native SHIM API interface provided by the socket layer. The layering model is illustrated in Figure 1. For simplicity, the IPsec layer has been excluded from the figure.



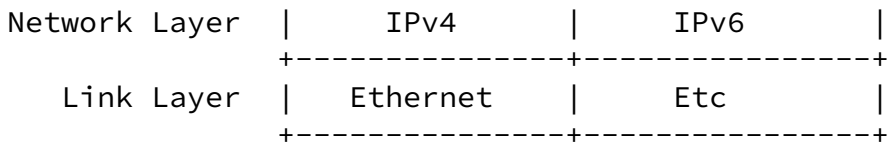


Figure 1

The SHIM layer is as a shim/wedge layer between the transport and network layers. The datagrams delivered between the transport and network layers are intercepted in the SHIM layer to see if the datagrams are SHIM related and require SHIM intervention.

3.2. Namespace Model

The namespace model is shown in Table 2 from SHIM point of view. The namespace identifiers are described in this section.

Layer	Identifier
User Interface	FQDN
Application Layer	ULID, port and protocol
Transport Layer	ULID, port
SHIM Layer	ULID
Network Layer	Locator

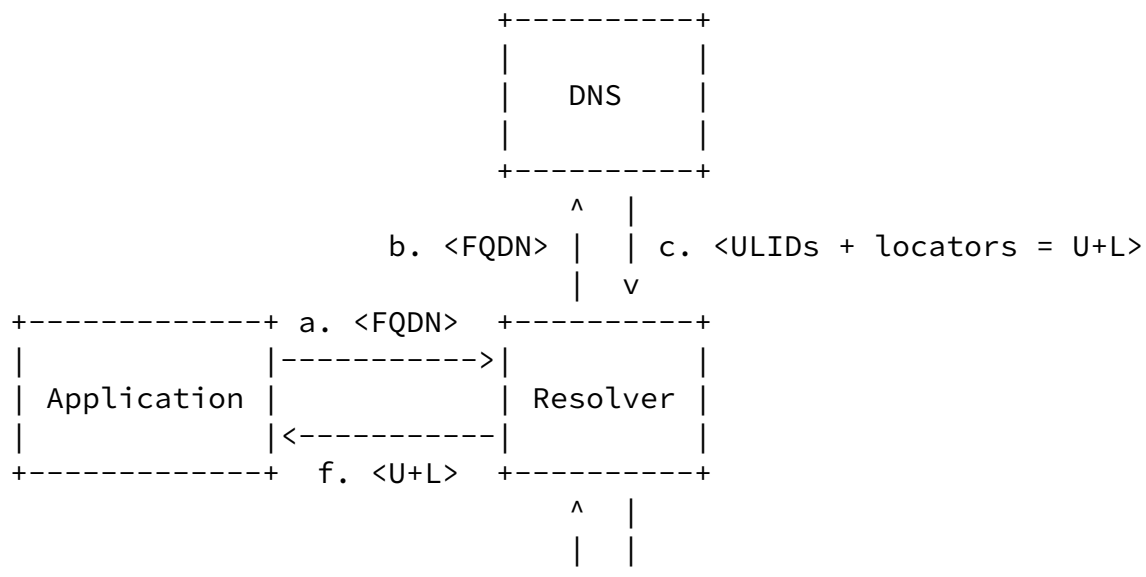
Table 2

User interfaces input human-readable names and translate them to machine-readable names. In SHIM API, the ULID is a HIT when the underlying protocol is HIP. The ULID is present at the application layer and also transport layer checksum is calculated based on it. The SHIM layer handles also ULIDs and translates them locators for the network layer.

3.3. Interaction with the Resolver

Before an application can establish network communications, it must

translate a hostname to the corresponding identifier(s). DNS based hostname-to-identifier translation is illustrated in Figure 2. The application calls the resolver (step a.) to resolve an FQDN (step b.). The DNS server responds with ULIDs and a set of locators (step c.). The resolver may not directly pass the ULIDs and the locators to the application, but may first inform to the SHIM module (steps d. and e.). Finally, the resolver passes the ULIDs and locators to the application (step f.).



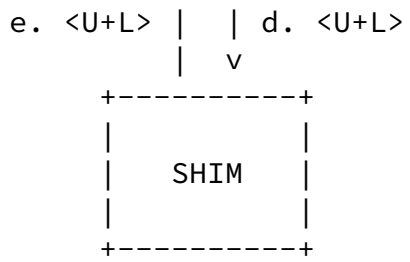


Figure 2

In practise, the resolver functionality can implemented in different ways. It may be implemented in existing resolver libraries or as an DNS proxy.

4. API Syntax and Semantics

In this section, we describe the native SHIM API using the syntax of the C programming language and present only the ``external'' interfaces and data structures that are visible to the applications. We limit the description to those interfaces and data structures that are either modified or completely new, because the SHIM API is otherwise identical to the sockets API [POSIX].

4.1. Socket Family and Address Structure

We introduce a new protocol family, PF_SHIM, for the sockets API. The AF_SHIM constant is an alias for it. The use of the PF_SHIM constant is mandatory with the socket() function if the SHIM API is to be used in the application. The PF_SHIM constant is given as the first argument (domain) to the socket() function.

The ULIDs and locators are contained in the sockaddr_shim structure,

which is shown in Figure 3. The family of the socket, shim_family, is set to PF_SHIM. The port number sins_port is two octets and the sins_ulid is four octets. The ULID value is an IPv6 address. The locator is an IPv6 address or IPv6-mapped address IPv4 address as defined in - [RFC3493]. The family is stored in host byte order and the ULID and locator are stored in network byte order.

```
typedef struct in6_addr shim_ulid_t;
```

```
typedef struct in6_addr shim_locator_t;

struct sockaddr_shim {
        uint8_t        sins_len;
        uint8_t        sins_family;
        uint16_t       sins_port;
        shim_ulid_t    sins_ulid;
        shim_locator_t sins_locator;
        uint64_t       sins_flags;
}
```

Figure 3

The application usually sets the sins_ulid field using the resolver. However, three special macros can be used to directly set a value into the sins_ulid field. The macros are SHIM_ANY, SHIM_ANY_PUB and SHIM_ANY_ANON. They denote an ULID value associated with a wildcard ULID of any, public, or anonymous type.

In server applications, the SHIM_* macros accept incoming connections to all of the ULID of the local host. The macros correspond to the sockets API macros INADDR_ANY and IN6ADDR_ANY_INIT, but they are applicable at the SHIM layer. It should be noticed that only one process at a time in a host can bind with the SHIM_*ANY macro to the same port to avoid ambiguous bindings.

A client application can use the SHIM_ANY macro to establish a connection when only the server locator (and not the ULID) is known beforehand.

In both client and server based applications, the use of the SHIM_ANY* macro accepts also non-SHIM based communications to maximize backwards compatibility. When the application wants enforces the use of SHIM-based communications with ORCHID prefix [RFC4843], it sets the flag SHIM_FLAG_ONLY_ORCHID in sins_flags. Alternatively, the application accepts both ORCHID and non-ORCHID-based communications, but informs the difference e.g. to the user. In this case, the application calls SHIM_IPV6_ADDR_IS_ORCHID macro which inputs a pointer to a in6_addr and returns 1 when the address has orchid prefix and 0 otherwise.

Applications can also implement access control using the ULIDs. In

such a case, the application can compare two ULIDs using memcmp() or similar function.

4.2. Resolver

The SHIM API uses the getaddrinfo resolver function which the application uses to query both ULIDs and locators. The resolver introduces a new data structure, which is used both as the input and output argument for the resolver. The data structure is illustrated in Figure 4.

```
struct addrinfo {
    int     ai_flags;          /* e.g. AI_SHIM */
    int     ai_family;         /* e.g. PF_SHIM */
    int     ai_socktype;       /* e.g. SOCK_STREAM */
    int     ai_protocol;       /* 0 or IPPROTO_HIP */
    size_t  ai_addrlen;        /* length of the endpoint */
    struct sockaddr *ai_addr; /* socket address */
    char    *ai_canonname;     /* canon. name of the host */
    struct addrinfo *ai_next; /* next endpoint */
};
```

Figure 4

In addrinfo structures, the family field is set to PF_SHIM when the socket address structure contains an ULID that refers to ULID, such as HIT.

The flag AI_SHIM must be set, or otherwise the resolver does not return sockaddr_shim data structures to guarantee that legacy applications do not break. Some applications may prefer configuring the locators manually and can set the AI_SHIM_NOLOCATORS flag to prohibit the getaddrinfo from resolving any locators.

The ai_family field is PF_SHIM with SHIM-specific addrinfo data structures.

The protocol field is 0 when the getaddrinfo caller does not care about the specific SHIM protocol to be used. The caller (or the resolver) can set this field also to IPPROTO_HIP.

ai_addrlen is the size of the structure pointed by ai_addr.

The ai_addr points to a sockaddr_shim structure when the value of ai_family is PF_SHIM. The resolver sets SHIM_RVS in sins_flags of the when the locator belongs to a rendezvous server [I-D.ietf-hip-dns].

The SHIM API does not introduce changes to the interface syntax of the existing sockets API functions, such as bind(), connect(), send(), sendto(), sendmsg(), recv(), recvfrom(), and recvmsg(). However, the SHIM-aware application usually passes the functions a sockaddr_shim structure instead of a sockaddr_in or sockaddr_in6 structure. A SHIM-aware application either creates the sockaddr_shim structures manually or obtains them from the resolver. The getaddrinfo resolver [RFC3493] is shown in Figure 5.

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

Figure 5

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. For example, the resolver returns only anonymous endpoints in the output argument res when the application sets the ai_addr pointer of hints to point to a sockaddr_shim structure with sins_ulid filled with SHIM_ANY_ANON. A NULL hints argument indicates that any kind of endpoints are acceptable.

The output argument res is dynamically allocated by the resolver. The application must free res argument with the free_addrinfo function. The res argument contains a linked list of the resolved endpoints. The linked list contains sockaddr_shim structures only when the input argument has the AI_SHIM flag set. When the resolver finds SHIM identifiers, it inserts them to the front of the list.

Resolving of a hostname may result in multiple locators associated to a single ULID, but the sockaddr_shim structure contains only a single ULID-locator pair. The resolver handles this by repeating the ULD as many time as needed. For example, let us consider a case where the resolver finds an ULID that is associated to two locators. In such a

case, the resolver outputs two sockaddr_shim structures with the same ULID but different locators.

5. IANA Considerations

No IANA considerations.

6. Security Considerations

To be done.

7. Acknowledgements

Jukka Ylitalo and Pekka Nikander have contributed many ideas, time and effort to the HIP API. Thomas Henderson, Kristian Slavov, Julien Laganier, Jaakko Kangasharju, Mika Kousa, Jan Melen, Andrew McGregor, Sasu Tarkoma, Lars Eggert, Joe Touch, Antti Jaervinen, Anthony Joseph, Teemu Koponen and Juha-Matti Tapio have also provided valuable ideas and feedback. Thanks for the APPS area folks, Stephane Bortzmeyer, Chris Newman, Tony Finch, "der Mouse" and especially Keith Moore for comments.

8. Normative References

[I-D.henderson-hip-applications]
Henderson, T. and P. Nikander, "Using HIP with Legacy Applications", draft-henderson-hip-applications-03 (work in progress), May 2006.

[I-D.ietf-btns-c-api]
Komu, M., "IPsec Application Programming Interfaces", draft-ietf-btns-c-api-00 (work in progress), June 2007.

[I-D.ietf-hip-dns]
Nikander, P. and J. Laganier, "Host Identity Protocol (HIP) Domain Name System (DNS) Extensions", draft-ietf-hip-dns-09 (work in progress), April 2007.

[I-D.ietf-shim6-multihome-shim-api]

Komu, M., "Socket Application Program Interface (API) for Multihoming Shim", draft-ietf-shim6-multihome-shim-api-02 (work in progress), March 2007.

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

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

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

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

Author's Address

Miika Komu
Helsinki Institute for Information Technology
Tammasaarenkatu 3
Helsinki
Finland

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

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information

on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).