Applying a Cryptographic Namespace to Applications

Miika Komu
Helsinki Institute for Information Technology
Advanced Research Unit
P.O. Box 9800
FIN-02015 HUT, Finland
miika@iki.fi

Sasu Tarkoma
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400
FIN-02015 HUT
sasu.tarkoma@tml.hut.fi

Jaakko Kangasharju
Helsinki Institute for Information Technology
Advanced Research Unit
P.O. Box 9800
FIN-02015 HUT, Finland
jkangash@hiit.fi

Andrei Gurtov
Helsinki Institute for Information Technology
Advanced Research Unit
P.O. Box 9800
FIN-02015 HUT, Finland
gurtov@cs.helsinki.fi

ABSTRACT

The Host Identity Protocol (HIP) is a promising solution for dynamic network interconnection. HIP introduces a namespace based on cryptographically generated Host Identifiers. In this paper, two different API variants for accessing the namespace are described, namely the legacy and the native APIs. Furthermore, we present our implementation experience on applying the APIs to a number of applications, including FTP, telnet, and personal mobility. Well-known problems of callbacks and referrals, i.e., passing the IP address within application messages, are considered for FTP in the context of HIP. We show that the callback problem is solvable using the legacy API. The APIs are important for easy transition to HIP-enabled networks. Our experimentation with well-known network applications indicate that porting applications to use the APIs is realistic.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*applications, protocol architecture*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation, Security, Standardization

Keywords

Host Identity Protocol, sockets API, referral, personal mobility

DIN'05, September 2, 2005, Cologne, Germany.

1. INTRODUCTION

The interconnection of mobile nodes, mobile networks, and multi-homed hosts is a challenging task within the current Internet architecture. The Host Identity Protocol (HIP) being developed by the IETF [8] is a promising solution to address these issues. The HIP layer is located between the network and transport layers and provides a new cryptographic addressing space for applications, where communication endpoints are identified using public cryptographic keys instead of IP addresses. However, HIP by itself provides no benefits unless there are applications using the protocol. In this paper, we consider the important problems of accommodating legacy applications to run on top of HIP, and of designing a native HIP API for new networking applications.

The fundamental idea behind HIP is to divide the address of a network-addressable node to two parts: the *identifier* and *locator* parts. The identifier part uniquely names the host using a cryptographic namespace and the locator part uniquely defines the topological location of the node in the network.

The main benefits of the new HIP namespace are statistically unique identifiers, separation of identifiers from their topological location, better support for delegation and intermediaries, multi-homing/mobility support, and security features such as authentication and confidentiality [8, 1, 15]. Recently, many systems based on globally unique flat namespaces have been proposed, including Unmanaged Internet Protocol (UIP) [3], i3 [15], and Delegation Oriented Architecture (DOA) [1]. IPv6 addresses some of the concerns with IPv4 and Network Address Translations (NATs), but still couples the identity of the hosts with the location.

The introduction of a new namespace requires consideration of two new architectural issues: how the new namespace is used in packets, and how the identifiers are resolved and distributed. In addition, mechanisms that allow applications to leverage the properties of the namespace are needed. We focus on the applications and present two APIs, *legacy API* [4] and *native HIP API* [6]. The APIs are all included in our HIP for Linux implementation [2].

The rest of the paper is organized as follows. A brief

overview of HIP architecture is given in Section 2. In Section 3, the legacy and native APIs for HIP are described. In Section 4, we illustrate the use of HIP APIs for FTP, Telnet, and personal mobility applications. Section 5 concludes the paper.

2. THE HIP NAMESPACE

HIP [8] introduces a new Host Identifier (HI) namespace for the Internet. The HIs are disjoint from the IPv4 and IPv6 namespaces in order to provide location independent identification of upper-layer endpoints. By decoupling the network-layer identifiers from the upper-layer identifiers, the HIP architecture provides a sound foundation on which to build mobility and multi-homing support. The upper layers have stable endpoint identifiers, but network-layer addresses can change dynamically.

The endpoints are identified using asymmetric cryptography. A HI is the public key component of an asymmetric key pair. The private key is owned by the endpoint, making impersonating another endpoint very difficult. HIP uses the HIs as Transport Layer Identifiers (TLIs). The locators, i.e., IP addresses, are used only in the network layer. There is a one-to-many binding between a HI and the corresponding locators [11]. As the HI is essentially a variable-sized public key, it is difficult to use in datagram headers. Further, long identifiers are difficult to support in the sockets API because it imposes a limit of 255 bytes to socket address structures. To address these problems, the HIP architecture also includes fixed-size representations of the HI. A Host Identity Tag (HIT) is a 128-bit long hash of the HI, and a Local Scope Identifier (LSI) is a 32-bit representation of the HIT.

In the traditional TCP/IP model, connection associations in the application layer are uniquely distinguished by the source IP address, destination IP address, source port, destination port, and transport protocol type. HIP changes this model by using HITs in the place of IP addresses. The HIP model is further expanded in the native HIP API model by using Endpoint Descriptors (EDs) instead of HITs. Now, the application layer uses source ED, destination ED, source port, destination port, and transport protocol type to distinguish between different connection associations. The namespace model used in the native HIP API is shown in Figure 1.

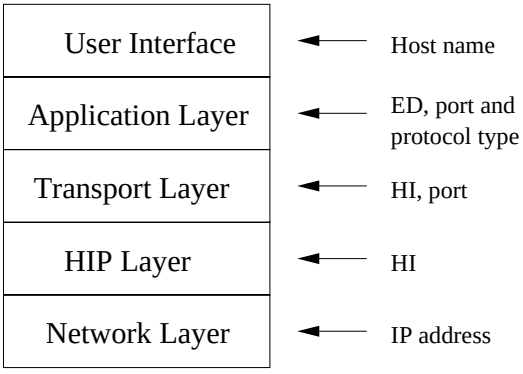

Figure 1: The HIP namespace model

The ED is used for hiding the representation of endpoints from applications in the native HIP API. It acts as a handle or an alias to the corresponding HI on the host. It is an integer having only local significance, similar to a file or socket descriptor. This kind of identifier with only local significance appears also in other namespace models, such as in OCALA [5].

The difference between the application and transport layer identifiers is that the transport layer uses HIs instead of EDs. The TLI is named with source HI, destination HI, source port, and destination port at the transport layer. Correspondingly, the HIP layer uses HIs as identifiers. The HIP Security Associations (SAs) are based on source HI and destination HI pairs. The network layer uses IP addresses, i.e., locators, for routing. The network layer interacts with the HIP layer to exchange information about changes in the addresses of local interfaces and peers.

The native HIP API socket bindings are visualized in Figure 2. A HIP socket is associated with one source and one destination ED, along with their port numbers and the protocol type. Multiple EDs and ports can be associated with a single HI. Further, the source HI is associated with a set of network interfaces at the local host. The destination HI, in turn, is associated with a set of destination addresses of the peer.

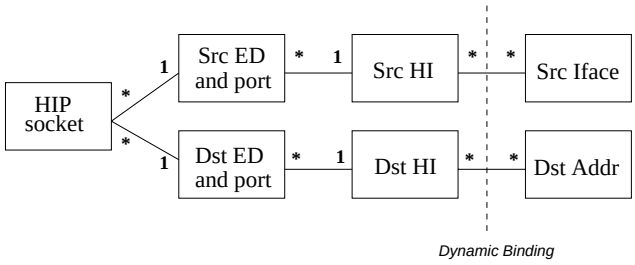

Figure 2: Native HIP API socket bindings

We believe that using EDs instead of HITs at the application layer has two useful properties. First, it simplifies implementing opportunistic base exchange, and second, EDs can be seen as a higher-layer concept to separate application-layer identifiers from those of lower layers.

In *opportunistic base exchange* the initiator does not know the responder's HIT, but only its IP address. Trying to implement this with the legacy API using the standard sockets API forces the application to associate its socket with the responder's IP address instead of its HIT. This increases the complexity of the HIP implementation, since a mapping from IPs to HITs is now needed, and it may not function reliably when IP addresses change due to mobility. In the native HIP API, however, the application binds to an ED. The HIP implementation can then transparently associate this ED with the responder's HIT that is learned later during the base exchange. Since EDs are already used in the native HIP API, supporting opportunistic mode does not increase the complexity of the HIP implementation.

It also seems to us that the new abstraction layer provided by the EDs may have some synergy with service identifiers [1] or session layer identifiers [14]. However, we are currently investigating this idea, and will not consider it further in this paper.

3. HIP API

In this section, we describe two APIs for applications to access the HIP namespace. The APIs are described in the C programming language, although Java is also supported by our implementation. First, we present the legacy API which is intended as an easy migration path towards HIP-enabled applications. Next, we present the native HIP API that allows applications to fully utilize the new namespace and protocol. Finally, we discuss problems related to referrals.

3.1 Legacy API

Network applications typically use host names to address peers. Host names have to be resolved to IPv6 addresses from the Domain Name System (DNS) in the resolver library before network connections can be established with peers. In the legacy API, the resolver routine has been modified to prefer HITs as the result of DNS queries instead of IPv6 addresses. Otherwise, the legacy API appears like the standard sockets API to the application.

We modified the resolver library to support HIP in two ways. In *transparent mode*, the DNS queries resolve silently to HITs instead of IPv6 addresses. For backwards compatibility, the resolver returns IP addresses if no HITs were found. The greatest benefit of the transparent mode is that it requires no changes in the application. However, a drawback of the transparent mode is that the resolver is not guaranteed to always return HITs. To address this shortcoming, applications can use the resolver in *explicit mode* by passing a flag explicitly to the resolver. This flag enforces the use of HIP by making the resolver return only HITs to the application. Effectively, this means that connections will be established using HIP or not at all. This way, HIP can be used with minimal changes in HIP-aware applications.

Example code using the legacy API is shown in Figure 3. The only modification from a standard socket application is the use of a flag flag to enable the explicit mode.

```
struct addrinfo hints, *res, *try;
char *hello = "hello";
int err, int bytes, sock;

memset(hints, 0, sizeof(hints));
hints.ai_flags = AI_HIP;
hints.ai_family = AF_INET6;
hints.ai_socktype = SOCK_STREAM;

err   = getaddrinfo("www.host.org", "echo",
                    &hints, &res);
sock  = socket(res->ai_family,
               res->ai_socktype,
               res->protocol);
for (try = res; try; try = try->ai_next)
  err = connect(sock, try->ai_addr,
                try->ai_addrlen);

bytes = send(sock, hello, strlen(hello), 0);
bytes = recv(sock, hello, strlen(hello), 0);
err   = close(sock);
err   = freeaddrinfo(res);
```

Figure 3: A "Hello, world" client using the legacy API.

3.2 Native HIP API

The legacy API requires only minor changes in applications, and therefore it cannot utilize all features of a HIP-enabled networking stack. Applications requiring more control over the HIP layer can use the native HIP API [6]. The most significant difference between the legacy and the native APIs is that the native HIP API can use public-key identities in the userspace sockets API. A direct benefit of this is that the users can provide their own public key identifiers to the networking stack. As a result, the identities are not bound to just hosts; they can be bound to users, processes, or groups. For instance, process migration systems [7] may benefit from this as the HI can be moved along with the process. In addition, if DNS is used to store public keys instead of HITs [10], the explicit public key handling in the native HIP API should become useful.

We propose a *PF_HIP* protocol family to be available in HIP-enabled network stacks. HIP-aware applications use the existing transport layer sockets API and specify this new protocol family when creating sockets. By creating a HIP-enabled socket, an application can detect whether HIP is supported on the local host. Similarly, an application can detect HIP support in a peer host by resolving the EDs of the peer. If the peer does not support HIP, the resolver returns an empty set.

The syntax of the native HIP API is similar to the legacy API. The crucial differences are the use of PF_HIP instead of AF_INET6, and a new socket structure for EDs. The resolver function is used in a similar way as the legacy API resolver [6]. An example use of the native HIP API is shown in Figure 4. The example uses an application-specified identifier from the file `/home/mk/hip_host_dsa_key`.

```
int sockfd, err, family = PF_HIP,
    type = SOCK_STREAM;
char *user_priv_key = "/home/mk/hip_host_dsa_key";
struct endpoint *endpoint;
struct sockaddr_ed my_ed;
struct endpointinfo hints, *res = NULL;

err = load_hip_endpoint_pem(user_priv_key,
                            &endpoint);
err = setmyeid(&my_ed, "", endpoint, NULL);
sockfd = socket(family, type, 0);
err = bind(sockfd, (struct sockaddr *) &my_ed,
           sizeof(my_ed));

memset(&hints, 0, sizeof(&hints));
hints.ei_socktype = type;
hints.ei_family = family;
err = getendpointinfo("www.host.org", "echo",
                      &hints, &res);

/* connect, send and recv as in Figure 3 */
```

Figure 4: A "Hello, world" client with application-specified identifiers in the native HIP API.

An application can control the HIP layer better using the native HIP API than the legacy API. For example, the application can set the base exchange puzzle to be more difficult for a specific server port number, request for higher SA lifetimes, use smaller (and less secure) key lengths, or

even specify its own HIs. Quality of Service (QoS) related attributes can also be accessed through the native HIP API to allow the simultaneous use of multiple IP flows. This enables applications to benefit from soft-handover strategies, or to select a data path depending on the available QoS. For example, the application can be notified when a LAN interface of the host is activated, so that the application can use it for data traffic instead of a slow WLAN link.

3.3 The Referral Problem

HIP introduces a new address space for the transport layer. Basically, the address space is flat although it is possible to use type 2 [8, 10] HITs that contain a domain prefix. Using the prefix, HITs can be resolved to IP addresses from the DNS. However, the problem with this approach is that it has some security implications due to the increased probability of HIT collisions. As a consequence, we may need to have full-length type 1 HITs [8, 10] in the future.

However, this causes problems for applications that need a remote application to initiate a connection. Currently they communicate either their own IP address (*callback*) or that of a third party (*referral*) [12]. The remote application will later connect to the communicated address. Using the legacy API with such applications would replace these IP addresses with HITs. However, since HITs cannot be resolved to IP addresses in the current DNS infrastructure, the remote application cannot typically initiate the required connection.

There are at least three ways to solve this problem. One way is to modify the DNS infrastructure to support type 2 HIT lookup. The second way is to use an overlay based on a flat namespace such as Internet Indirection Infrastructure (i3) [17] to support resolving of HITs. Third, the overlay can also be used for packet routing, at least for the initial HIP signaling [9], but this is out of the scope of this paper.

4. HIP APPLICATIONS

In this section, we present three HIP-enabled applications. We begin with an FTP application, which has been labeled by the community as challenging for HIP. Then, we examine a Telnet application that was ported to use the native HIP API. Finally, we describe an application of personal mobility with HIP.

4.1 FTP and Referrals

File Transfer Protocol (FTP) [13] uses two separate channels (TCP connections) for communication, one for control and one for data. The data channel can be initiated in two ways. In a passive mode, the server passes its IP address and port number as a callback to the client using the control channel. Then the client initiates a data channel to the server based on the IP address and port number given by the server. In an active mode, it is the vice versa: the server initiates the data channel to the IP address and port given by the client.

The FTP way of passing addresses as callbacks can be considered problematic when HIP is used because HITs are used instead of IP addresses. The crux of the problem is that the application may not be able to resolve a given HIT to a routable IP address. We decided to experiment with this problem using the legacy API with an initial expectation of failure.

Our callback experiment used IPv6-enabled FTP client and server software (lftp version 3.1.3 and proftpd version 1.2.10). Both the client and the server used the legacy API, thus requiring no modifications. We carried out a simple test where the client contacted the HIT of the server and downloaded a file. Surprisingly, both modes, active and passive, worked properly. We also experimented with a mobility handover during file download by changing the currently active address of the client. This caused only a relatively small delay during the file download.

The reason why the callback worked in the case of FTP is that once the client (initiator) and server (responder) have established a security association, they are aware of each other's HIT-to-IP mappings. The mapping from the HIT to IP address(es) is not lost because it is valid at least for the lifetime of the IPsec SA.

However, the callbacks are only a part of the problem. In the FTP case, it is possible to use referrals instead of callbacks, but fortunately this feature is rarely used. The referral problem occurs when the client creates a new data channel using the FTP protocol to server A, but redirects it to another server B. As the redirection is based on a HIT, server B must resolve the HIT to an IP address, which requires support from the infrastructure. We already described three general solutions to this problem in Section 3.3. Additionally, it would also be possible to extend the FTP protocol to use either FQDNs or HITs and IP addresses together.

4.2 Telnet

We ported an IPv6-enabled Telnet client and daemon to use the native HIP API. We configured the native HIP API into the code as a compile-time option. The porting process itself was quite straightforward. As the native HIP API resolver name and related data structure are named differently from their IP-based API correspondents, the porting process consisted mainly of search and replace operations in the source code. The API names are different to emphasize the introduction of the new namespace in the resolver but the syntax is almost identical [6].

4.3 Personal Mobility

Personal mobility and device personalization are becoming an important part of applications and mass-market devices. Personal mobility occurs when the user changes devices. Personalization is needed to change the user experience on a new device to meet the user's expectations. Almost all recent mobile phones support personalization of the device to accommodate the user's preferences, for example in call settings, buddy-lists, and user interface appearance.

A HI can be used to support personal mobility and device personalization. This is accomplished by associating the HI with a user and using a smartcard or a USB stick to store the HI. Personal mobility takes place when the user inserts the identity storage device containing the HI into a terminal device. The HI can then be used to initiate a HIP connection, and to support mobility and multi-homing. The HI may also be used to locate, download, and synchronize data needed for device personalization, such as device, user interface, and preference profiles. Since the HI is a public cryptographic key, it allows authentication of the client as well as confidentiality of the personalization data.

We experimented with a scenario in which the HI is stored on a USB memory stick and can be moved between different machines. The insertion of the USB stick is detected

automatically. After detection, the HI is loaded to the HIP kernel module, and then used by applications. We demonstrated HIP-based connections in personal mobility for file synchronization and for streaming audio playback.

In our scenario, the USB stick only contains the HI, and the connections are not persistent. In the future we also plan to store data related to the session state on the stick so that connections can be restored at the new location. In addition, we envision that the USB stick can be replaced with a smart card that can create and verify signatures directly. This way, users can use their personal identities even on untrusted hosts (for example in Internet cafes) without compromising their private keys. Further, users can prevent other people from tracking their personal identifier and location by using either short-lived HIs or "blinded" HITs [16].

5. CONCLUSION

In this paper we have presented the legacy and the native APIs of our Linux-based HIP implementation. The legacy API does not necessarily require changes to applications. The native API requires modifications, but allows applications to provide their own public key identities. The cryptographic namespace is useful for applications and we discussed the implications for three example applications: FTP, Telnet, and personal mobility and personalization. Initially, we expected HIP-enabled FTP to be hindered by the callback problem, but our analysis and experimentation showed that callbacks are not an issue. We observed that it was relatively straightforward to port a Telnet utility to use the native HIP API. As an example of the benefits of the native HIP API, we discussed personal mobility and device personalization using Host Identifiers and USB sticks.

As a conclusion, we envisage that simple network applications use the legacy API in a transparent fashion, and more advanced applications utilize the new namespace using the native HIP API. We expect that the migration to HIP may require changes in some applications, but our experiments with basic networking utilities and the legacy API indicate that the introduction of the new namespace is realistic.

6. ACKNOWLEDGMENTS

We thank Jukka Ylitalo, Jeff Ahrenholz, Teemu Koponen, Abhinav Pathak and Thomas Henderson for their comments on the the paper. Niklas Karlsson developed a small feature to the HIP module that was required for the FTP daemon to work (binding to a HIT using the legacy API). We also thank the anonymous reviewers for their helpful comments.

7. REFERENCES

[1] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *Proc. of ACM SIGCOMM'04*, pages 343–352, Aug. 2004.

[2] C. Candolin, M. Komu, M. Kousa, and J. Lundberg. An implementation of HIP for Linux. In *Proc. of the Linux Symposium*, July 2003.

[3] B. Ford. Unmanaged Internet Protocol: taming the edge network management crisis. *ACM Computer Communication Review*, 34(1):93–98, 2004.

[4] T. R. Henderson. Using HIP with legacy applications: draft-henderson-hip-applications-01, July 2005. Work in progress. Expires in January 19, 2006.

[5] J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. Supporting legacy applications over i3. Technical Report UCB/CSD-04-1342, University of California at Berkeley, May 2004.

[6] M. Komu. *Native Application Programming Interfaces for the Host Identity Protocol: draft-mkomu-hip-native-api-00*. Internet Engineering Task Force, Sept. 2004. Work in progress. Expires August, 2005.

[7] T. Koponen, A. Gurtov, and P. Nikander. Application mobility with Host Identity Protocol. In *Proc. of NDSS Wireless and Security Workshop*, San Diego, CA, USA, Feb. 2005. Internet Society.

[8] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host Identity Protocol: draft-ietf-hip-base-03, June. 2005. Work in progress. Expires in December, 2005.

[9] P. Nikander, J. Arkko, and B. Ohlman. Host Identity Indirection Infrastructure (Hi3). In *Proc. of The Second Swedish National Computer Networking Workshop 2004 (SNCNW2004)*, Karlstad, Sweden, Nov. 2004.

[10] P. Nikander and J. Laganier. Host Identity Protocol (HIP) Domain Name System (DNS) extensions: draft-ietf-hip-dns-01.txt, Feb. 2005. Work in progress. Expires in August, 2005.

[11] P. Nikander, J. Ylitalo, and J. Wall. Integrating Security, Mobility, and Multi-homing in a HIP way. In *Proc. of Network and Distributed Systems Security Symposium (NDSS'03)*, San Diego, CA, USA, Feb. 2003. Internet Society.

[12] E. Nordmark. *Multi6 Application Referral Issues*. Internet Engineering Task Force, Jan. 2005. Internet draft, work in progress.

[13] J. Postel and J. Reynolds. *RFC959: File Transfer Protocol (FTP)*. Internet Engineering Task Force, Oct. 1985.

[14] M. A. C. Snoeren. *A Session-based Architecture for Internet Mobility*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Feb. 2003.

[15] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proc. of ACM SIGCOMM'02*, Pittsburgh, PA, USA, Aug. 2002.

[16] J. Ylitalo and P. Nikander. BLIND: A complete identity protection framework for end-points. In *Proc. of the Twelfth International Workshop on Security Protocols*, Apr. 2004.

[17] S. Zhuang, K. Lai, I. Stoica, R. Katz, and S. Shenker. Host Mobility using an Internet Indirection Infrastructure. Technical report, University of California at Berkeley, 2002.