An Implementation of HIP for Linux

Miika Komu[*]
Miika.Komu@hiit.fi

*Mika Kousa**
Mika.Kousa@hiit.fi

Janne Lundberg[†]
Janne.Lundberg@hut.fi

Catharina Candolin [‡]
Catharina.Candolin@hut.fi

Abstract

One of the main problems with IP has been its lack of security. Although IPSec and DNSSec have provided some level of security to IP, the notion of a true identity for hosts is still missing. Typically, the IP address of the host has been used as the host identity, regardless of the fact that it is nothing more than routing information. The purpose of the Host Identity Payload/Protocol (HIP) architecture is to add a cryptographically based name space, the Host Identity, to the IP protocol. The Host Identity serves as the identity of the host, whereas the IP address is merely used for routing purposes. In this paper, we describe the HIP architecture further, and present our IPv6 based implementation of HIP for Linux.

1 Introduction

The lack of security has been one of the main problems with IP. Although IPSec [8] and DNSSec [14] have provided some level of security to IP, such as data origin authentication, confidentiality, integrity, and so forth, the notion of a true identity for hosts is still missing. The IP address has typically been used both to identify the host and to provide routing information. This has led to the misuse of IP addresses for identification purposes in many security schemes. To overcome the problems related to the current use of IP addresses, the Host Identity Payload/Protocol (HIP) architecture adds a cryptographically based name space, the Host Identity, to the IP protocol. Each host (or more specifically, its networking kernel or stack) is assigned at least one Host Identity, which can be either public or anonymous. The Host Identity can be used for authentication purposes to support trust between systems, enhance mobility and dynamic IP renumbering, aid in protocol translation/transition and reduce denial-of-service attacks. Furthermore, as all of the higher protocols are bound to the Host Identity instead of the IP address, the IP address can now be used solely for routing purposes.

In this paper, we describe the HIP architecture and present our IPv6 [6] based implementation of HIP for Linux. The rest of the paper is structured as follows: in Section 2, the HIP architecture and the Host Layer Protocol is described. Section 3 describes our implementation, and Section 4 concludes the paper.

[*]Helsinki Institute for Information Technology, P.O. Box 9800, FIN-02015 HUT, Finland

[†]Laboratory for Theoretical Computer Science, Helsinki University of Technology, P.O. Box 9205, FIN-02015 HUT, Finland

[‡]Laboratory for Theoretical Computer Science, Helsinki University of Technology, P.O. Box 5400, FIN-02015 HUT, Finland

2 The HIP architecture

There are two name spaces in use in the Internet today: IP addresses and domain names. IP addresses have been used both to identify the network interface of the host and the routing direction vector. The three main problems with the current name spaces are that dynamic readdressing cannot be directly managed, anonymity is not provided in a consistent and trustable manner, and authentication for systems and datagrams is not provided.

In [10][11][12], the HIP architecture is introduced. HIP introduces a new cryptographically based name space, the Host Identity (HI), and adds a Host Layer between the network and the transport layer in the IP stack.

The modification to the IP stack is depicted in Figure 1. In the current architecture, each process is identified by a process ID (PID). The process may establish transport layer connections to other hosts (or to the host itself), and the transport layer connection is then identified using the source and destination IP addresses as well as the source and destination ports. On the IP layer, the IP address is used as the endpoint identifier, and on the MAC layer, the hardware address is used. In HIP, the transport layer is modified so that the connections are identified using the source and destination HIs as well as the source and destination ports. HIP then provides a binding between the HIs and the IP addresses, e.g. using DNS [9].

The HI is typically a cryptographic public key, which serves as the endpoint identifier of the node. Each host will have at least one HI assigned to its networking kernel or stack. The HI can be either public or anonymous. Public HIs may be stored in directories, such as DNS, in order to allow the host to be contacted by other hosts. A host may have several HIs, and it may also generate temporary (anonymous)

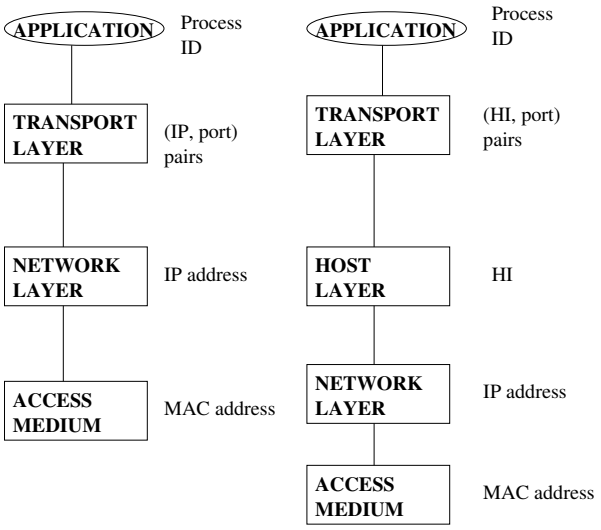

Figure 1: The current IP stack and the HIP based stack

HIs on the fly for establishing connections to other hosts. The main purpose of anonymous HIs is to provide privacy protection to the host, should the host not wish to use its public HI(s).

The HI is never directly used in any Internet protocol. It is stored in a repository, and is passed in HIP. Protocols use a 128-bit Host Identity Tag (HIT), which is a hash of the HI. Another representation of the HI is the Local Scope Identity (LSI), which has a size of 32 bits, but is local to the host. Its main purpose is to support backwards compatibility with the IPv4 API.

The main advantages of using HIT in protocols instead of the HI is that its fixed length makes protocol coding easier and also does not add as much overhead to the data packets as a public key would. It also presents a consistent format to the protocol regardless of the underlying identity technology used. HIT functions much like the SPI does in IPSec, but instead of being an arbitrary 32-bit value that identifies the Security Association for a datagram (together with the destination IP address and security protocol), HIT identifies the public key

that can validate the packet authentication.

The probability that a collision will occur is extremely small. However, should there be two public keys for one HIT, the HIT acts as a hint for the correct public key to use.

The HIP architecture basically solves the problems of dynamic readdressing, anonymity, and authentication. As the IP address no longer functions as an endpoint identifier, the problem of mobility becomes trivial, as the node may easily change its HI and IP address bindings as it moves. Anonymity is provided by temporary and anonymous HIs. Furthermore, as the name space is cryptographically based, it becomes possible to perform authentication based on the HIs. In [13], the concept of integrating security, mobility, and multi-homing based on HIP is discussed further.

2.1 The Host Layer Protocol

The Host Layer Protocol (HLP) is a signaling protocol between the communicating endpoints. The main purpose of the protocol is to perform mutual end-to-end authentication and to create IPSec ESP [7] Security Associations to be used for integrity protection and possibly also encryption. Furthermore, the protocol performs reachability verification using a simple challenge-response scheme.

The HLP protocol provides seven message types, of which four are dedicated to the base exchange. In Figure 2, the base exchange is depicted. In the first message, $I1$, the initiator I sends its own HIT and the HIT of the responder to the responder. The responder R replies with message $R1$, which contains the HITs of I and itself as well as a puzzle based challenge for I to solve. The purpose of the challenge is to make the protocol resistant to denial-of-service attacks. (Puzzle based schemes have been previously used for providing DoS protection to

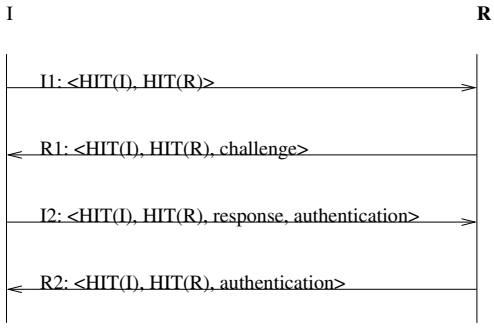

Figure 2: The base exchange of the Host Layer Protocol

both authentication [3] and encryption [5] protocols.) I solves the puzzle and sends in $I2$ the HITs of itself and R as well as the solution to the puzzle, and performs the authentication. $R2$ now commits itself to the communication, and responds with the HITs of I and itself, and performs the authentication. After this, I and R have performed the mutual authentication and established Security Associations for ESP, and can now engage in secure communications. Furthermore, reachability is verified by the fact that the protocol has more than two rounds.

If I does not have any prior information of R, it may retrieve the information from a repository, such as DNS. I sends a lookup query to the DNS server, which replies with R's address, HI, and HIT.

There are three other messages in the HLP. The HIP New SPI Packet (NES) provides the peer system with its new SPI, provides a new Diffie-Hellman key to produce new keying material, and provides any intermediate system with the mapping of the old SPI to the new. The HIP Readdress Packet (REA) allows a host to notify its partners of a change of the IP address (e.g. as a result of mobility). The HIP Bootstrap Packet (BOS) is used when the initiator is unable to learn a responders information from a repository.

3 Implementation

The protocol is implemented as a kernel module which uses a user space daemon process for some cryptographic operations, such as computation and verification of DSA signatures. Since the protocol is implemented as a kernel module, the kernel can remain as intact as possible, with only minor modifications. The modifications are backwards compatible so that normal TCP/IP connectivity without HIP can still be used. The state information required by the protocol state machine is located within the kernel module. The user space daemon acts as a slave to the kernel module and does not contain state.

The implementation is based on the Linux kernel version 2.4.18 with USAGI [2] patches. The implementation supports HIP only over IPv6.

3.1 Network Socket API

The most important design goal in the network socket API has been the transparent use of HIP by legacy applications. Thus, the legacy applications do not need any changes in their source code to utilize the benefits of HIP. On the other hand, applications that are HIP aware should be able to perform some additional tasks that will not be available to legacy applications. For example, a HIP aware application may require or deny the use of HIP. A reason to require HIP would be to benefit from the multihoming, security, and mobility features of HIP. A reason to deny the use of HIP might be to avoid the extra overhead caused by the cryptographic operations in a device with limited computing capacity.

A typical network application does not usually establish a network connection directly to an IPv6 address. Instead, the application is usually given the hostname of the peer, which has to be resolved to an IPv6 address from DNS. The connection can then be established to the IPv6 address.

When HIP is used, the network application needs additional support in the resolver for two different reasons. The first reason is that the resolver should return HITs instead of IPv6 addresses if HIP is being used transparently in a legacy application. The second reason is that a HIT to IPv6 address mapping should always be sent to the kernel as a side effect of the domain name query. Otherwise the IPv6 layer in the kernel does not have an address it can use for routing packets.

The resolver interfaces are traditionally contained in `libc`. The USAGI project has its own modified version of `libc` which is also used in the implementation. Only the `getaddrinfo` resolver interface is currently supported in the implementation for experimentation purposes.

Most of the legacy IPv6 applications, such as telnet clients and web browsers, are able to use HIP in transparent mode if they can access the HIP enabled resolver. This means that they should be relinked against the HIP patched USAGI `libc`. Firewalls, network address translators and other applications that handle raw packets may need changes in application code in order to utilize HIP.

3.2 Userspace daemon

A userspace daemon is required by the HIP module for several reasons, of which the most important reason is that the protocol requires DSA and Diffie-Hellman cryptographic algorithms which cannot easily be implemented within the kernel. Unfortunately, there are no known kernel cryptographic libraries supporting those algorithms, so those tasks have to be done in user space libraries. The HIPL im-

plementation uses the OpenSSL [1] library for user space cryptographic operations.

The daemon is used by the kernel module to perform many small operations on data. The kernel module can send queries such as *sign the given data with this DSA key* or *solve this cookie* to the daemon. The daemon calculates a response for the given query and the response either contains the answer to the query or an error message if something went wrong. Messages between the kernel module and the daemon are exchanged synchronously in a request-response fashion.

The request-response communication is implemented using a common interface in the kernel. When a request is carried out in the kernel, the current context of execution will be saved. The contents of the context depends on the operation being executed, but a minimal context defines at least a reference to a callback function. The callback function is called after the request has been served in the daemon and the daemon has sent a response message to the kernel module. The kernel module can restore its state based on the information stored in the context and then continue its execution where it left off.

The actual request-response communication is implemented in a straightforward manner. The implementation can serve only one daemon request at a time, and subsequent requests are saved into a FIFO queue. An arriving response message from the daemon triggers a new daemon request from the top of the FIFO queue.

3.3 Networking Stack

Transport layer communications are bound to HITs when HIP is used. When data is sent over the transport layer connections, packets are created and received as if they were using HITs as the source and destination addresses in the transport layer headers. As the packets are passed up and down the protocol stack, they will encounter a number of hooks that may intercept the passing packet to the HIP module for modifications. Currently, the implementation has three major entry points into the HIP module from the IPv6 stack.

IPv6 output functions. The hooks in the output functions are triggered after the packet has been built and the packet has passed IPsec ESP processing. If the packet belongs to a HIP connection, it has a HIT instead of an IPv6 address as the destination address in the IPv6 header. Such a packet will be intercepted by the HIP module. Other packets are allowed through intact.

When the HIP output functions receive a packet, the module first checks whether the packet belongs to an established HIP connection by searching its table of established connections using the source and destination HITs as the key. If an existing connection is not found, the packet is dropped and a HIP exchange is started by sending an I1 packet. On the other hand, if an existing connection is found, the source and destination HITs in the IPv6 header are replaced by the IPv6 addresses that are stored in the mapping table in the kernel. Thus, even if connections are maintained using HITs as identifiers in the transport layer, the actual packets that are sent to the network will still always contain valid IPv6 addresses.

IPv6 ESP input functions. All received packets that belong to an established HIP connection will have an ESP header. Therefore, it is only necessary to intercept HIP packets from within ESP. Packets that are received by ESP are classified to those that belong to a HIP connection and those that do not. The reverse of the output mapping is performed. The

correct mapping is located using the SPI field in the ESP header, and if a mapping is found, the source and destination addresses in the packet are replaced by the corresponding HITs before the packet is forwarded to the actual ESP processing. Again, the ESP processing only sees the HITs, and not the IPv6 addresses.

HIP protocol input. A special case is required for HIP packets that are received during the connection establishment phase. The HIP module is registered as a transport layer protocol and does not actually require a special hook for this functionality. This intercept point is used to receive I1, R1, I2 and R2 packets, as well as other HIP negotiation packets.

Also, a few other hooks are required in order to, for example, make the neighbor discovery in IPv6 work correctly with HIP. However, these hooks are not relevant for this discussion.

3.4 Collaboration of Components

This section gives an overview of the collaboration of the components through an example. Figure 3 represents an overview of the system architecture and the logical connections between the components.

For simplicity, only a minimalistic base exchange is demonstrated. For example, mobility and multihoming are not demonstrated here. The configuration in this example consists of two hosts which use legacy applications that have not been designed for HIP, and therefore the transparent mode is used. The host that starts the HIP exchange will be referred to as the initiator while the peer is known as the responder. The initiator is a host that wishes to browse a web page from another host, and the responder has a web server listening for incoming requests. A DNS server in the domain of

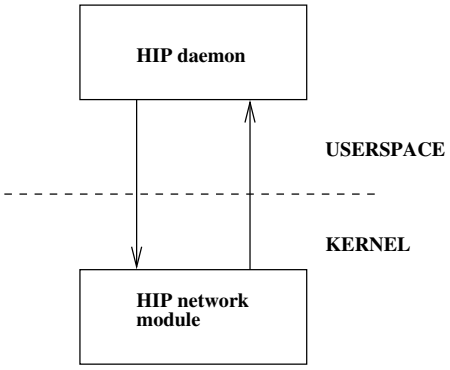

Figure 3: Collaboration of components

the responder is configured to return the IPv6 address and the HIT of the responder.

When the HIP module is loaded into the kernel, it first queries a Host Identity from the daemon. This identity will be used in all signed HIP packets that are sent by the host. The kernel module also generates a list of prebuilt R1 packets for quick sending. Finally, the module registers its hooks into the kernel. HIP connections can now be established.

When the user of the initiator host inputs the URL of the requested web page into the web browser to view the web pages in the responder's web server, the browser queries DNS for the name of the host using the resolver routine `getaddrinfo`. Since the browser is linked to the modified `libinet6`, the query is handled by the modified resolver.

The resolver queries the DNS for the responder's hostname. When the resolver receives the response from the DNS, it finds IPv6 addresses as well as HITs in the reply. Two things are done before the DNS reply is returned in a list from the resolver to the web browser. First, the resolver changes the order of the addresses in the DNS reply list before returning them to the application. The HITs of the responder are placed in the beginning of the list before the IPv6 addresses of the responder. Second, the

kernel is notified about the mapping of the HIT to the corresponding IPv6 address.

The result of the resolver call is a list containing first the HIT and then the IPv6 address of the responder. The browser is assumed to make the straightforward choice and select the first address from the list, which is a HIT in this case. The HIT is then used in the socket calls. Because the browser uses the HTTP [4] protocol which runs over TCP, the browser passes the HIT to the TCP `connect` call in the network socket API.

The `connect` call is handled by the TCP layer in the kernel. The TCP layer begins a handshake to establish a connection by generating a SYN packet to be delivered to the web server. When the SYN packet is encapsulated into an IPv6 packet in the IPv6 layer, the packet is captured by the output hook of the HIP module. The HIP module then examines the packet and discovers that the addresses in the packets are HITs instead of regular IPv6 addresses. The module also attempts to lookup a previously established HIP connection from its table of established HIP connections. The lookup fails because the connection attempt was a new one and a HIP exchange is needed to establish the new HIP connection. Since a HIP connection between the client and the server does not exist, the TCP SYN packet cannot immediately be delivered to the web server. For simplicity, the SYN packet will be dropped. Retransmissions will also be dropped until the base exchange has been completed.

To start the base exchange, the initiator sends an I1 packet to the web server. R1, I2 and R2 packets are exchanged after this. The building and parsing of each of the R1, I2, and R2 packets requires the assistance of the HIP daemon. For example, the daemon verifies the validity of the identity of the peer from DNS and creates a symmetric Diffie-Hellman key for the hosts during the base exchange.

Once the base exchange is completed, the hosts will have generated a common secret that they will be able to use to secure their communication. They will also have established IPsec Security Associations that will be used to encrypt the communication between the hosts. The TCP handshake can continue, and once it has been completed, the initiator can receive web pages from the web server at the responder. If further TCP connections need to be established between the two hosts, the HIP negotiation is not needed to be performed again, but the existing security associations are reused for the new connections.

4 Conclusion

In this paper, we described the HIP architecture, which has been designed to overcome problems mainly with respect to security, mobility, and privacy in the current Internet. HIP adds a new layer, the Host Layer, between the networking and transport layer in the IP stack, and introduces a Host Identity (HI) to serve as an end-point identifier of the host. Typically, the HI is represented by a public key. Each host will have at least one HI assigned to its networking kernel or stack. As the HI is used to identify the hosts, the IP addresses are used merely for routing purposes.

HIP defines a Host Layer Protocol to be used as a signaling protocol between end hosts. The purpose of the protocol is to perform mutual end-to-end authentication and to establish IPSec Security Associations. HLP consists of seven message types, of which four are part of the HIP base exchange.

As part of this paper, we presented our IPv6 based implementation of HIP for Linux. The Host Layer Protocol is implemented as a kernel module, which uses a user space daemon

process to perform some cryptographic operations. The advantage of our approach is that the kernel can remain as intact as possible, with only minor modifications. Furthermore, the modifications are backwards compatible so that the host is able to do networking without HIP. Our implementation is based on Linux kernel version 2.4.18 with USAGI patches.

Acknowledgments

This research is funded by the National Technology Agency of Finland (TEKES), Elisa Communications, Ericsson, Nokia, TeliaSonera, Creanor, and More Magic Software. We thank Jukka Ylitalo, Jorma Wall, Petri Jokela, and especially Pekka Nikander from Ericsson Research for fruitful cooperation and interoperability testing with their implementation of HIP for BSD. Furthermore, we are grateful for the valuable discussions we have had with Andrew McGregor and Thomas Henderson.

References

[1] Openssl: The open source toolkit for ssl/tls. `http://www.openssl.org/`.

[2] Usagi project. `http://www.linux-ipv6.org/`.

[3] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles,.

[4] T. Berners-Lee, R. Fielding, H. Frystyk, J. Gettys, and J. Mogul. Hypertext Transfer Protocol – HTTP/1.1. Technical report, Internet Engineering Task Force, January 1997. RFC 2068.

[5] Catharina Candolin, Janne Lundberg, and Pekka Nikander. Experimenting with early opportunistic key agreement. In *Proceedings of Workshop SEcurity of Communication on Internet, Internet Communication Security*, Tunis, Tunisia, September 2002.

[6] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. IETF Request for Comments 2460, December 1998.

[7] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). Request for Comments 2406, 1998.

[8] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. IETF Request for Comments 2401, 1998.

[9] P. Mockapetris. Domain Names — Implementation and Specification. IETF RFC 1035, 1987.

[10] R. Moskowitz. Host identity payload and protocol. Internet Draft draft-moskowitz-hip-05.txt, work in progress, 2001.

[11] R. Moskowitz. Host identity payload architecture. Internet Draft draft-ietf-moskowitz-hip-arch-02.txt, work in progress, 2001.

[12] R. Moskowitz. Host Identity Payload Implementation. Internet Draft draft-ietf-moskowitz-hip-impl-01.txt, work in progress, 2001.

[13] P. Nikander, J. Ylitalo, and J Wall. Integrating Security, Mobility, and Multi-homing in a HIP way. In *Proceedings of Network and Distributed Systems Security Symposium (NDSS'03)*, pages 87–99, San Diego, USA, February 2003.

[14] B. Wellington. Domain Name System Security (DNSSEC) Signing Authority.

IETF Request for Comments 3008,
November 2000.

Proceedings of the Linux Symposium

July 23th–26th, 2003
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Alan Cox, *Red Hat, Inc.*
Andi Kleen, *SuSE, GmbH*
Matthew Wilcox, *Hewlett-Packard*
Gerrit Huizenga, *IBM*
Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Martin K. Petersen, *Wild Open Source, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*