HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory

Host Identity Protocol

Privacy Management

Master's Thesis

Laura Takkinen

Telecommunications Software and Multimedia Laboratory
Espoo 2006

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Computer Science Degree Program

ABSTRACT OF MASTER'S THESIS

Author: Laura Takkinen

Title of thesis:
Host Identity Protocol Privacy Management

Date: March 30 2006 **Pages:** 11 + 70
Professorship: Telecommunications software **Code:** T-110
Supervisor: Professor Antti Ylä-Jääski
Instructor: Janne Lindqvist, M.Sc. (Tech.)

The Host Identity Protocol (HIP) proposes a solution to many problems of the current Internet. HIP improves the security of communication by providing a mutual authentication of the communicating peers. HIP is used together with the Encapsulating Security Payload (ESP), which offers encryption services for the upper layer protocols.

Despite of many useful security facilities of HIP, it does not support anonymity and location privacy of the peers. The HIP privacy management is an application, which purpose is to solve the privacy problems of the HIP protocol. The HIP privacy management allows user to choose what kind of identifiers HIP-aware applications are allowed to use.

The basic idea of the HIP privacy management is to force all HIP-aware applications to use random identifiers in the link layer, network layer and HIP layer. In addition, the HIP privacy management forces applications to change the link layer MAC addresses and network layer IP addresses periodically.

This thesis presents the design, implementation and analysis of the HIP privacy management prototype. The prototype does not solve the whole privacy problem of HIP instead, it solves pseudonymity and location privacy problems. However, the prototype can be extended to support complete privacy protection.

Keywords: HIP privacy management, anonymity, location privacy, link layer, network layer, HIP layer, IPv6 address, MAC address
Language: English

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Tietotekniikan osasto
Tietotekniikan koulutusohjelma

Tekijä:	Laura Takkinen	
Työn nimi:		
Host Identity Protocol ja Yksityisyyden Hallinta		
Päiväys:	30. maaliskuuta 2006	**Sivumäärä:** 11 + 70
Professuuri:	Tietoliikenneohjelmistot	**Koodi:** T-110
Työn valvoja:	Professori Antti Ylä-Jääski	
Työn ohjaaja:	DI Janne Lindqvist	

Host Identity Protocol (HIP) on uusi tietoliikenneprotokolla, joka tarjoaa ratkaisun moniin nykyisessä Internetissä esiintyviin ongelmiin. HIP-protokolla parantaa tietoliikenteen turvallisuutta mahdollistamalla yhteyden osapuolten keskinäisen todennuksen. HIP-protokollaa käytetään yhdessä Encapsulating Security Payload (ESP) -protokollan kanssa, joka mahdollistaa tiedonsiirron salauksen.

Huolimatta kaikista HIP-protokollan tietoturvallisuutta parantavista ominaisuuksista, se ei kuitenkaan takaa käyttäjän anonymiteettiä eikä sijainnin yksityisyyttä. Tämä diplomityö esittelee sovelluksen, jonka tarkoituksena on tarjota HIP-protokollaa käyttäville järjestelmille yksityisyyden hallintapalvelu. Palvelu mahdollistaa sen, että järjestelmän käyttäjä voi kontrolloida millaisia tunnisteita HIP-sovellukset saavat käyttää.

Yksityisyyden hallinnan perusajatuksena on, että HIP-sovelluksia pakotetaan käyttämään satunnaisia tunnisteita kolmella eri tasolla: linkkitasolla, verkkotasolla ja HIP-tasolla. Lisäksi linkkitason tunnisteita eli MAC-osoitteita ja verkkotason tunnisteita eli IP-osoitteita vaihdellaan säännöllisin väliajoin.

Diplomityö esittelee prototyypin arkkitehtuurin, toteutuksen ja analysoi lopputulosta. Toteutettu prototyyppi ei ratkaise HIP-protokollan yksityisyysongelmaa kokonaan, vaan se ratkaisee pseudonymiteetin ja sijainnin yksityisyyden ongelmat. Prototyyppi tarjoaa kuitenkin hyvän lähtökohdan HIP-protokollan yksityisyyden hallinnan jatkokehittämiselle.

Avainsanat:	HIP, yksityisyyden hallinta, anonymiteetti, sijainnin yksityisyys, linkkitaso, verkkotaso, HIP-taso, IPv6-osoite, MAC-osoite
Kieli:	Englanti

Acknowledgements

This thesis was written as a part of the InfraHIP project, which is a cooperation project of Helsinki Institute of Information Technology and Helsinki University of Technology.

I would like to thank my supervisor Professor Antti Ylä-Jääski for his guidance, usefull comments and writing tips. I also want to thank my instructor Janne Lindqvist whose constant feedback and patience helped me through the writing process.

I want to thank Miika Komu for giving me expertise help with HIPL. The discussions we had cleared my head many times and gave me many useful ideas. I wish to thank Juha-Matti Tapio for giving me useful C programming tips and answering my numerous Linux questions. He also implemented a great test application, which is presented in the Analysis Chapter 7.

Sincere thanks to all the people in the InfraHIP project and my colleagues in the TML laboratory for making such a nice work environment. It has been a pleasure to work with you all.

Last but not least I want to thank my dear life-companion Harri for beeing there for me and supporting me in my everyday life. I want greatly thank my parents for all love and encouragement they have given me.

Espoo March 30th 2006

Laura Takkinen

Abbreviations and Acronyms

API	Application Programming Interface
AR	Access Router
DHCPv6	Dynamic Host Configuration Protocol for IPv6
DNS	Domain Name System
DoS	Denial of Service
ED	Endpoint Descriptor
ESP	Encapsulating Security Payload
EUI-64	64-bit Extended Unique Identifier
FQDN	Fully Qualified Domain Name
GUI	Graphical User Interface
HI	Host Identifier
HIP	Host Identity Protocol
HIPL	HIP for Linux
HIT	Host Identity Tag
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPsec	Internet Protocol security
LAN	Local Area Network
MAC	Media Access Control
MitM	Man in the Middle
OUI	Organizationally Unique Identifier

PID	Process Identifier
PKI	Public Key Infrastructure
PPID	Parent Process Identifier
RVA	Rendezvous Agent
SA	Security Association
SID	Session Identifier
SPI	Security Parameter Index
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Communication technology has evolved rapidly during the last decade. Computers and the underlying network technology have become more efficient and cheaper allowing more people to join the public Internet with broadband connections.

The current trends of communication technology such as widespread mobile technology, wireless communication, high demands for Quality of Service (QoS) and security have brought along new kinds of requirements for the underlying communication software. This thesis concentrates on improving the security of a communication software, or more precisely *privacy*. Privacy is one of the most fundamental security requirements for network services today.

The current naming convention of the Internet does not support well security and mobility needs of modern networking applications. For instance, authentication of the communicating parties cannot be provided directly, and anonymity and dynamic readdressing both need separate solutions [18]. Dynamic readdressing is complicated to accomplish because IP addresses have currently dual role: they are used as *locators* and *identifiers* [19]. An IP address determines the topological location of the host and is also used as a unique name of the host.

The Host Identity Protocol (HIP) [18, 19] proposes solution to many problems of the current Internet. Basically the protocol creates a secure communication channel between two parties called peers. In addition, it provides many useful features such as mutual authentication of the peers, integrity protection of the payload and support for dynamic readdressing.

The HIP communication context called a HIP association, is created through

a base exchange, which is a cryptographic protocol based on the authenticated Diffie-Hellman key exchange [19]. The base exchange allows peers to authenticate themselves to each other, and it also provides resistance against Denial-of-Service (DoS) and Man-in-the-Middle (MitM) attacks [18]. Durign the base exchange peers negotiate security association (SA) for Encapsulating Security Payload (ESP) transport format [23]. The HIP protocol is used with this cryptographic protocol, which provides encryption services for upper layer protocols such as the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

Currently, when two peers initiate a transport layer association, TCP connection or UDP association, it is bound to the IP addresses. HIP renews this traditional and unflexible naming convention by introducing the Host Identity namespace between the transport and network layers of the TCP/IP stack. The new namespace consists of Host Identifiers (HI) that are public-key identifiers. A host may have several HIs. Some of the HIs are public and could be published, for instance, in the Domain Name System (DNS), or in some Public Key Infrastructure (PKI). The host can also have anonymous HIs that are not published.

In the HIP protocol, the transport layer association is bound to the HIs [18] of the peers. This new naming mechanism frees IP addresses from their dual role as locators and identifiers. When using HIP, IP addresses can work purely as locators, and connections are identified with HIs. Releasing IP addresses from their identifier role facilitates the implementation of mobility. IP addresses are free to change during the communication because they indicate only location information. The HIP protocol supports readdressing by providing an update mechanism for HIP associations [23]. With this mechanism the HIP associations can be updated in such way that IP addresses can be changed without breaking the connections.

1.1 Problem Statement

Although HIP includes many useful security features, it does not offer full privacy protection to the communicating peers. HIP provides one solution for anonymity, but it does not solve the problem completely: it provides anonymity only on the HIP layer allowing an initiator of a HIP connection to use an anonymous HI as a source HI. The reason why anonymity is not complete is that link layer identifiers (Media Access Control address, MAC) and network layer identifiers (IPv6 address) are disclosed in every packet sent or received by the host. This provides an efficient way to trace the network

activity or location of the host.

It must be noted here that there are also other privacy related problems than anonymity with HIP communication. Linkability is one of them. Generally, linkability means that an attacker is able to bind together two or more messages, actions or events. For a user point of view, this means that if he is using same services and resources multiple times, the attacker can link these network activities together.

In practice, the HIP traffic can be linked together by following the Security Parameter Indexes (SPI) carried in the ESP headers. SPIs are used to map the encrypted data packets to the correct HIP association [19]. In addition, MAC and IPv6 addresses do not only break the anonymity of the host, but they also link together the network traffic of the host.

To protect the identity and location of a host, the host must use private identifiers in all layers of the HIP networking stack: anonymous HIs, random IP addresses and random MAC addresses. Also, individual applications should not be able to choose which type of identifiers they use. If one networking application uses public identifiers and the other applications anonymous identifiers, the identity of the host can be compromised.

The word random in the context of IPv6 addresses is somewhat misleading. IPv6 addresses cannot be totally random because they must provide some topology information for routers to be able to route the packets to the correct responder. An IPv6 enabled host can configure its IPv6 address either from a DHCPv6 server, or by itself using a feature called the *stateless address autoconfiguration* [27]. In the stateless address autoconfiguration an IPv6 address is generated by combining the network identifier of the subnet with an interface identifier of the host. The interface identifier is generated from the link layer identifier (MAC) of the host. This basically means that when the location of the host changes, only the network part of the address changes, and host part remains unchanged. This feature of IPv6 provides eavesdroppers a way to trace the network behaviour and movements of mobile hosts.

One solution to this IP-level privacy problem is the *privacy extension for stateless address autoconfiguration* [20]. It suggests that instead of using addresses generated via stateless address autoconfiguration, referred to as *normal* IPv6 addresses, the user should be able to choose whether to use so called *temporary* IPv6 addresses to communication. Temporary IPv6 addresses differ from normal IPv6 addresses that they contain randomly generated interface identifiers instead of MAC based identifiers. Moreover, RFC 3041 [20] defines that these temporary addresses are changed periodically to make tracing more difficult.

To summarize, the privacy problem is not limited to a single networking layer and should not be solved independently on any of the layers. The first problem concerns HIP layer identifiers, which are associated with a communication endpoint. The second problem concerns IPv6 and MAC addresses, and the third one concerns the interdependency between these two. For instance, replacing the IPv6 address of a network device with a temporary one will not be enough if the MAC address remains unchanged. This means that if the IPv6 address changes, the MAC must also be changed. Any change or update of the IPv6 address and MAC address must be performed in a synchronized way. If only one layer identifier is changed, the privacy can be compromised. [10]

1.2 Scope

The goal of this thesis is to design and implement a proof-of-concept prototype of the privacy management for HIP. The purpose of the prototype is to provide partial solution for the privacy problem of the HIP protocol. This thesis provides solutions for the pseudonymity and location privacy problems. The pseudonymity means that the true identity of the host cannot be revealed, but the host can be linked to the certain network activities. The location privacy means an ability to hide the current and past locations of the host.

The prototype solves these problems by forcing all HIP-aware networking applications of the host to use randomly generated identifiers in three layers of the HIP enabled TCP/IP stack (HIP, network and link layers). Solving the other privacy related problems of HIP, such as linkability and anonymity, is out of the scope of this thesis.

The prototype is built on top of the HIP and IPv6 protocols. The prototype cannot be used to protect IPv6 traffic only, it requires the usage of the HIP protocol on top of the IPv6 protocol. In addition, the prototype is limited to support only the IPv6 protocol and not IPv4. Underlying network technology constrains the scope as well, the prototype supports only Ethernet technology according to the IEEE standard 802-2001 [26].

Chapter 2

Background

This chapter explains briefly the privacy related terminology and discusses the problems that the HIP privacy management solves. Underlying technology for the prototype is also described. This includes the next generation Internet protocol IPv6 and HIP protocol. One essential feature of IPv6 is the stateless address autoconfiguration, which is explained in more detail later in this chapter. This feature plays a fundamental role in the implementation of the HIP privacy management. The chapter describes also basic functionality of the HIP protocol: base exchange, new namespace and how the protocol solves the problems of mobility and multi-homing.

2.1 What Is Privacy?

Privacy is a basic human right. Several definitions have been proposed for privacy. For example, according to Alan Westin [31] privacy is: "the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others".

Privacy in the context of networking involves many different aspects. Privacy can be for instance, the use of some technique to protect IP addresses, MAC addresses, location of the host, content of the messages, context of the communication and so on [9].

According to an Internet Draft [9] privacy aspects can be classified into the following four categories:

- Anonymity
- Unlinkability

- Unobservability
- Pseudonymity

Anonymity in the context of this thesis means that if identifiers such as HITs, IP addresses or MACs are used, neither the communicating peer nor any third party should be able to reveal the link between the used identifiers and the real identity of the user.

Unlinkability means that an attacker is not able to bind together for example messages, actions or events. From a user point of view he can use same services and resources multiple times without nobody being able to link the activities together.

Unobservability is a state where messages, actions or events cannot be distinguished from any other comparable target. For a user this means that he can use a particular service or resource without any third party being able to observe that the service or resource is being used.

Pseudonymity is weaker than anonymity. It means that a third party cannot identify a node, but may observe that two pseudonym actions were performed by the same node. For a user pseudonymity means that he can use resources or services without reveiling his identity, but he can still be responsible for that use.

2.1.1 Location Privacy

Location privacy means that third parties can trace neither the current nor past location of the host. The location means in this context topological location. To protect the location privacy, user must hide any relation between his location information and identification information. [9]

2.1.2 Whar Problems Are Solved?

This thesis solves partly the problems of pseudonymity and location privacy. The reason why pseudonymity problem is solved instead of anonymity is that after the HIP association is created, the anonymous HI, used in the base exchange, is no longer anonymous. It is known by the peer and possibly by some third parties as well. They cannot identify the host, but they know that the host that owns this certain HI is responsible for some network activities. Pseudonymity is reached by forcing networking applications to use private

identifiers in three networking layers. These layer identifiers are anonymous HIs, randomized IPv6 addresses and MAC addresses.

To extend the solution to solve the anonymity as well, there should be a way to change the HITs of the association such as MACs and IPv6 addresses are changed. However, this is out of the scope of this thesis.

Location privacy cannot be protected completely with the method presented in this thesis. Network prefixes of the IPv6 addresses reveal the current subnet in which the host is located. If the host is the only host of the subnet, the network prefix reveals its exact location.

Rest of the thesis uses simply the term privacy to refer both pseudonymity and location privacy.

2.2 Internet Protocol version 6

The IPv6 protocol preserves many features of the IPv4 protocol, but reshapes most of the protocol details. It introduces many new features, but only the ones that are essential for the HIP privacy management are presented in this thesis. The most important feature for the HIP privacy management is the IPv6 addressing scheme because the implementation of the prototype exploits it. Especially important is the stateless address autoconfiguration that is discussed in more detail in the following section.

2.2.1 IPv6 Addressing

Besides reshaping the whole IP datagram format, the IPv6 protocol represents a larger address space than IPv4. IPv6 addresses are 128 bits long instead of 32 bits of IPv4 addresses and are categorized into the following three classes [11]:

- *Unicast*: Identifies a single interface. If it is used as a destination address, packets are delivered to the interface identified by the unicast address.
- *Anycast*: Identifies a set of interfaces. If it is used as a destination address, packets are delivered to one of the interfaces identified by the anycast address.

Allocation	Format Prefix	Fraction of Address Space
Aggregatable Global Unicast Addresses	001	1/8
Link-Local Unicast Addresses	1111 1110 10	1/1024
Multicast Addresses	1111 1111	1/256

Table 2.1: Different types of IPv6 addresses and format prefixes [11].

- *Multicast*: Identifies a set of interfaces. If it is used as a destination address, packets are delivered to all interfaces identified by the multicast address.

IPv6 unicast addresses can further be classified as *local* or *global* scope addresses. Local unicast addresses enable communication between hosts within the same physical network. This kind of scenario solves two problems. Firstly, datagrams are not forwarded across the Internet, they stay inside the subnet. Secondly, hosts of an isolated network are able to communicate with each other without any connection to the public Internet.

Global scope unicast addresses are used to communication across the public Internet, and they are also used within the HIP privacy management. Interfaces of the host must have at least one link-local unicast address, but they can also have multiple addresses of any class (unicast, anycast and multicast) and scope (local, global) [11].

There was also a third scope of addresses, *site local*, but it was deprecated because of the ambiguity of addresses and unclear definition of the site. RFC 3879 [12] deprecates officially the site local addresses.

There exists different types of IPv6 addresses as indicated above. The type of the address can be recognized by the leading bits of it, called *format prefix*. Table 2.1 shows some examples of different IPv6 address types and the corresponding format prefixes.

An IPv6 unicast address can roughly be devided into a subnet prefix and host suffix, also called as *interface identifier* (see figure 2.1). Addresses with format prefixes 001 through 111 (except multicast addresses) all require 64-bit Extended Unique Identifier (EUI-64) format interface identifiers.

In the case of global unicast addresses, network prefix consists of global format prefix, routing prefix and subnet prefix (together 64 bits). The host

suffix is 64-bit, EUI-64 presentation of the link layer identifier. In the case of local unicast addresses the subnet prefix is replaced by the well-known link-local prefix FE80::0 (1111 1110 10 in binary format).

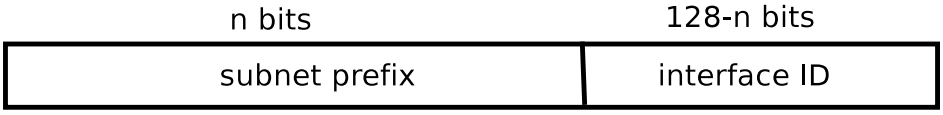

Figure 2.1: The structure of the IPv6 address.

Interface Identifiers

When a host uses the stateless address autoconfiguration to generate its IPv6 address, the link layer identifier of the host's network device is encoded and used as a host suffix of the address. The host suffix identifies the interface on that specific link. Naturally the host may have several network devices, and each device has its own MAC and IPv6 addresses.

The interface identifier of the host must be unique on the link, but it can be also globally unique. The same interface identifier can be used on multiple interfaces on a single host [11].

There are several types of different link layer addresses depending on the underlying network technology. Probably the most well know link layer identifier is 48-bit IEEE 802 identifier [26], which is used, for instance, in Ethernet technology. To ensure interoperability between different types of network technologies, the IPv6 protocol determines encoding for hardware addresses.

In the case of certain format prefixes, the interface identifier portion of an IPv6 address is required to be 64-bit, EUI-64 format identifier defined by IEEE standard [13]. The implementation presented in this thesis supports only Ethernet technology, which means that when a 48-bit IEEE 802 identifier is available, it must be used to create the EUI-64 identifier.

Figure 2.2 shows an example of a MAC address where the first three octets (c) indicate Organizationally Unique Identifier (OUI) assigned by IEEE, and the bits marked as m form a vendor supplied extension. The bit marked as zero is the *universal/local bit*, which indicates that the hardware address is assigned universally. If it is set to 1, it means that the whole hardware address (48 bits) has been assigned by a local administrator. The bit marked as g is the *group/individual bit*. If the bit is assigned to 0, it means that the address belongs to an individual host of the network. The value 1 indicates

that the address identifies a group of the hosts (or all hosts) of some Local Area Network (LAN) [26].

The HIP privacy management prototype uses randomly generated MAC addresses that are marked as locally administered individual addresses (first byte of the MAC address is cccccc10).

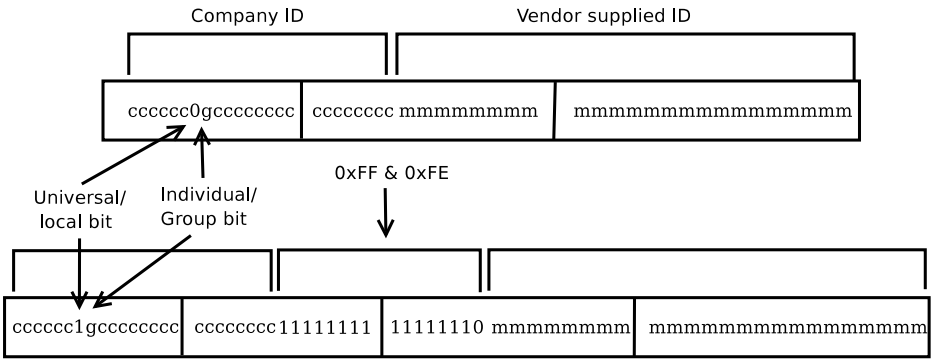

Figure 2.2: Converting IEEE 802 identifier to EUI-64 identifier [11].

When a 64-bit EUI-64 identifier is created from a 48-bit MAC identifier, two octets of bits (0xFF = 11111111 and 0xFE = 11111110) are added between the OUI and vendor identifier. Also the universal/local bit is inverted.

Stateless Address Autoconfiguration

The IPv6 protocol has a feature called the stateless address autoconfiguration that is defined by RFC 2462 [27]. This feature enables that all multicast-capable hosts of any network can start to communicate without any manual configuration of IPv6 addresses or presence of a DHCPv6 server.

By using the stateless address autoconfiguration, a host can generate link-local and global scope IPv6 addresses from the network prefix and its own interface identifier. To be able to create the global scope address, there must be an IPv6 router present in the network advertising the global scope network prefixes.

Autoconfiguration takes place during a system startup. At the beginning, host generates its link-local address as described earlier. Sending a Neighbor Solicitation message (defined by RFC 2461 [22] about the Neighbor Discovery Protocol), host can verify that the generated link-local address is unique on that link.

When an IP-level connectivity with the neighboring hosts is obtained, the host begins to listen Router Advertisement messages and possibly sends a Router Solicitation message to obtain the advertisements more quickly.

If routers are present, they send Router Advertisement messages containing information about what kind of autoconfiguration hosts should do. For instance, whether they should use the *stateful autoconfiguration* to obtain some additional configuration information. The stateful mechanism allows hosts to obtain their configuration information from a server.

Router Advertisement messages can also contain information about network prefixes and their lifetimes. Hosts can use this information to create statelessly global unicast addresses. To keep the prefix and other information up to date, the hosts refresh the information according to the Router Advertisement messages that routers send periodically.

An IPv6 address has a lifetime, which indicates how long it remains valid on a certain network interface. Basically, the IPv6 address is leased to a network interface for a certain period of time. After the lifetime expires, the address is removed from the network device.

The address goes through a couple of steps before it becomes invalid. At the beginning the address is preferred and can be used to initiate new connections. After some time it becomes deprecated indicating that the address binding will soon become invalid. Ongoing connections may still use the deprecated address, but new connections should use the preferred address. After the deprecated phase, the address will become invalid and is removed from the network interface.

The address leasing facilitates site renumbering by offering a mechanism to change addresses gracefully. The network layer provides a service for upper layers to select the most suitable (and preferred) source address for given destination address. However, applications may use the services of the socket Application Programming Interfaces (APIs) [7] to select the source addresses by themselves when initiating new connections. [27]

Privacy Extension for Stateless Address Autoconfiguration

The stateless address autoconfiguration generates global scope unicast addresses by combining global scope subnet prefixes with EUI-64 interface identifiers. If available, the interface identifiers are generated from the embedded IEEE identifiers of the network devices. That is, the interface identifier portion remains fixed even if the subnet prefix of the address changes. If

addresses with the same interface identifier are used constanly in multiple contexts, a malicious network user can keep track of network activity of a victim host [21].

The privacy extension for stateless address autoconfiguration [20], simply referred to as *privacy extension*, enables that hosts that use stateless address autoconfiguration can choose different IPv6 addresses for different communication contexts. The solution is to generate additional IPv6 addresses, where static interface identifiers are replaced with randomly generated, periodically changing sequence of bytes [20].

The privacy extension presents two new terms for addresses: *public* and *temporary* addresses. The term public refers to the normal IPv6 address that is gererated through the stateless address autoconfiguration and includes the MAC based EUI-64 format interface identifier. On the contrary, temporary address means an address, which interface identifier is generated randomly. Word temporary refers to the fact that the address is changed periodically. Or more precisely, the interface identifier part is changed periodically, the subnet prefix changes only when its lifetime expires.

Temporary addresses have a corresponding public address. When a host receives a Router Advertisement message including the subnet prefix, it generates a global unicast address from the prefix. Simultaneously, it generates also a new temporary address by using that same prefix.

Network devices can be configured to prefer the usage of temporary unicast addresses as a source address over the public ones. Chapter 3.1.1 discusses in more detail how this configuration can be done.

Temporary IPv6 addresses can be used to initiate connections. Addresses have a lifetime facility similar with normal IPv6 addresses. When initiating a connection and choosing a source address, the preferred temporary address must be used. Deprecated addresses may be used with open connections, but not to initiatiate any new connections. After the deprecated address expires, it is removed from the network interface.

When a temporary address becomes deprecated new temporary address is generated to replace it. Lifetimes of temporary addresses are determined by the lifetime values of corresponding public address, or values that user can determine. Everytime a host receives a Router Advertisement message and refreshes the address information of the public addresses according to the message, it also refreshes the information of corresponding temporary addresses [20].

2.3 Host Identity Protocol

IP addresses have two different roles in the current TCP/IP technology, they are used as locators that provide routing information for packets and identifiers that identify communicating endpoints. In other words, the location of the host also identifies the host. In the past, IP numbers had long-term significance, that is, computers had only one static IP address. A natural design choise was to identify transport layer connections with IP addresses [32].

The nature of network usage has changed since this design choise was made. For instance, there are increasing amount of mobile devices and hosts with multiple network interfaces (multi-homed hosts) using network services today. The main problem with the current naming convention is that it supports neither dynamic readdressing nor provides anonymity or authentication of the systems or datagrams [18].

The HIP protocol proposes one solution to these problems. HIP is designed as an end-to-end authentication and key establishment protocol, and it is used with the IPsec Encapsulating Security Payload (ESP) protocol. ESP provides integrity protection for the HIP payload [19].

HIP renews the current TCP/IP architecture by introducing a new, cryptographic namespace, Host Identity namespace, between the transport and network layers (see figure 2.3). The new namespace relieves IP addresses from their dual role as locators and identifiers. It consists of cryptographic names, called Host Identifiers (HI), that are public keys of asymmetric key pairs. HIs are used to identify the communicating endpoints allowing IP addresses to work purely as locators.

2.3.1 Protocol Description

HIP describes a base exchange, which is a two-party cryptographic protocol to create a secure communication context called a HIP association, between two hosts (see figure 2.4). The base exchange consists of four messages, where the initiator of the base exchange and the responder change Diffie-Hellman keys and authenticate themselves to each other.

The base exchange uses a puzzle mechanism to prevent denial of service attacks (DoS) towards responder. The responder sends a puzzle to the initiator in the second message delaying the HIP association creation. The initiator must solve the puzzle before the base exchange may continue. Durign the

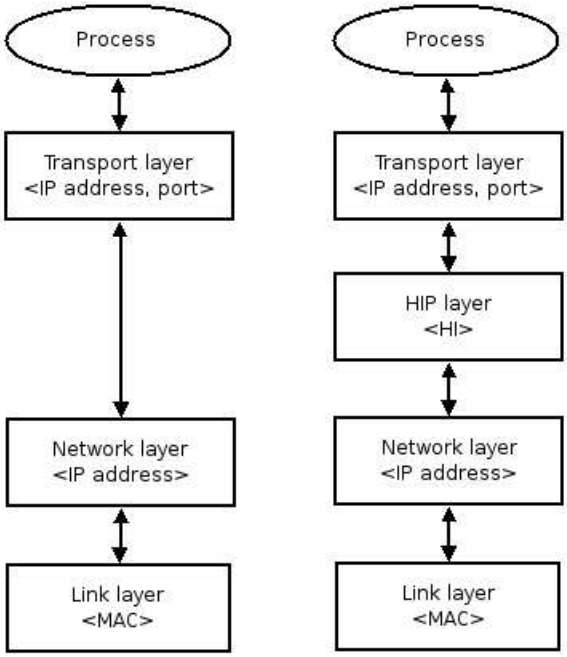

Figure 2.3: The current and HIP networking architectures [25].

base exchange peers negotiate security associations (SA) for ESP transport format [23]. An ESP secured HIP payload can start to flow after the fourth message [19].

2.3.2 Host Identifiers

HI is a public key of an asymmetric key pair. There exists different public key algorithms with different key lengths. It may cause problems if the different length HIs were used in packet headers and index values of tables.

The HIP protocol represents a fixed length (128-bit) presentation of HI, Host Identity Tag (HIT), which is a hashed enconding of HI. The advantage of using HITs instead of HIs, is that those are same size as IPv6 addresses and can therefore be used to replace IPv6 address fields in APIs and other protocols [19].

A host can have multiple identities, some of them *public* and some *anonymous*. Public identifiers can be stored in the Domain Name System (DNS) or some Public Key Infrastructure (PKI). Anonymous HIs are not stored

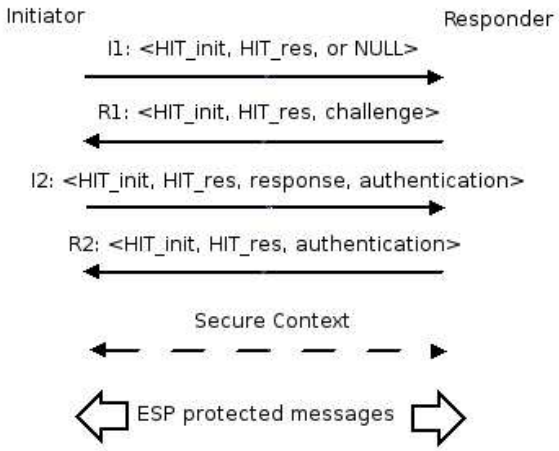

Figure 2.4: HIP base exchange messages.

anywhere, they are known only by the host itself [18]. The HIP privacy management provides a way for a host to force HIP-aware applications to use only anonymous HIs to communication.

2.3.3 Mobility and Multi-homing

The HIP protocol extension for end-host mobility and multi-homing is determined by the HIP working group's Internet-Draft [23]. The extension defines a parameter, called `LOCATOR`, which host can use in HIP messages to inform its peers about alternate addresses at which it can be reached.

The reason why host may have to change its address can be, for instance, network renumbering, a new DHCP lease or actual movement to another location in the Internet [23]. The HIP privacy management causes also periodical address changes.

When a mobile host changes its IP address, it must inform its peers about the change. The host sends an `UPDATE` message containing the `LOCATOR` parameter. For example, the host moves to another location and it needs to change its IPv6 address. It sends an `UPDATE` message to its peer. The message contains a `LOCATOR` parameter, which includes the new IPv6 address. The host decides to not to rekey its security association (SA) and formats the `ESP_INFO` parameter of the `UPDATE` message accordingly. The peer verifies the new IPv6 address by sending a response to the mobile hosts new address.

It also updates the address bindings of its local HIP associations. Once the mobile host responds to the peer's verification message, the peer marks the new address as `ACTIVE` and removes the old one. Both the mobile host or its peers may decide to rekey they security associations (SA) during the update procedure. Rekeying can be triggered by using the `ESP_INFO` parameter of the `UPDATE` message. [23]

A multi-homing host has several network interfaces. The host can inform its peers about the available addresses and which one is preferred address by using the `UPDATE` message including the `LOCATOR` parameter. By default, the preferred address is the one that was used in the base exchange. [23]

2.4 Related Work

There exists some related work to protect the privacy of communicating endpoints. BLIND [33] is an identity protection framework. It presents a privacy enhanced Diffie-Hellman protocol that provides identity protection for communicating parties. The protocol is completed with forwarding agents to ensure location privacy of the endpoints.

The framework contains a renewed TCP/IP architecture with additional layer between the network and transport layers. The framework uses cryptographic namespace, for instance, Host Identity namespace to separate the locator and identifier roles of location names [33].

The BLIND together with forwarding agents provides a complete identity and location protection of communicating parties. They obtain the complete identity privacy without knowing the location of each other. Only what is required is that the initiator must know the hash of the public key of its peer. The public keys that are used to identify the endpoins are blinded in such way that it is impossible to guess the keys. The locators are hided using the forwarding agents [33].

As a comparison, the HIP privacy management protects only the initiator's identity. Also, it provides partial location privacy without any third party services. The location is protected by making possible the use of randomized IPv6 addresses and link layer addresses. The protection is only partial because the IPv6 addresses contain real subnet prefixes that reveal some location information.

Another solution suggests the use of pseudo-random number sequences to replace the identifiers in the protocol stack and other trackable values [3].

The basic idea is that peers agree on a number of pseudo-random number sequences while they initiate their communication context. If each of the peers wants to protect its privacy, it uses these pseudo-random numbers to hide all trackable information of the outgoing traffic [3].

This method goes further than the HIP privacy management in that it changes all possible values: layer identifiers and all other tracable values. Moreover, the communicating peers must agree to use the method, whereas the usage of the HIP privacy management depends only on the initiating party. In addition, this method does not require any changes to the TCP/IP architecture.

The HIP location privacy extension [17] is a HIP-specific framework to provide location privacy and mobility for HIP-aware hosts. The proposed solution concentrates on the network layer problems. The framework requires two additional functional entities called Rendezvous Agent (RVA) and Access Router (AR). RVA provides location privacy to its attendants, it handles HIT-IP resolution and hides the locators of outgoing packets of mobile HIP host and incoming packets of its peers. The basic idea of the extension is that the HIT-IP resolution is delegated into the RVA. This delegation causes that inside a specific network section, called RVA protected area, locators are protected or not used at all. HITs are used to identify the traffic path inside this area. [17]

RVA protected areas are separated from the Internet by RVAs. The areas consists of several ARs, that are directly connected to some RVA. Mobile hosts are under the ARs. An AR keeps a HIT based neighbor list of all mobile nodes under it. ARs advertise also RVA advertisement messages, that mobile hosts can use to learn the HIT of the local RVA. [17]

The location privacy framework differs from the HIP privacy management in several ways. First of all, the framework provides only network layer location privacy. The MAC addresses can be used to trace the HIP traffic. Also, the framework requires a special network architecture with two additional network entities. The HIP privacy management does not require any special network topology or services of third parties. It is simply a software that can be run by arbitrary HIP host. The framework requires also some changes to the basic HIP mechanism [17]. The HIP privacy management does not alter the existing HIP protocol, it adds a new functionality to the existing protocol.

2.5 Summary

To summarize, the HIP privacy management provides a partial solution to the privacy problem of the HIP protocol. It solves the problems of pseudonymity and location privacy. The IPv6 and HIP protocols define features that can be used to implement the privacy management, for instance, the stateless address autoconfiguration, which provides a ready technique to generate temporary IPv6 addresses. In addition, mobility support of the HIP protocol enables that IPv6 addresses of the HIP host may be changed without termination of the active HIP association. The HIP protocol allows also the use of anonymous HIs when initializing the HIP association.

Chapter 3

Linux Background

This chapter discusses in more detail about the techniques that are used to implement the HIP privacy management. Emphasis is on the IPv6 and HIP protocol implementations on Linux. Only the issues that are relevant to the privacy management are covered. The chapter also presents some Linux specific techniques, for instance the Linux daemon implementation.

3.1 IPv6 in Linux

The IPv6 protocol implementation is inside the Linux kernel. The whole protocol functionality is located in directory `net/ipv6` under the Linux source tree. The chapter concentrates on the IPv6 addressing scheme and privacy extension that are defined in a single file called `addrconf.c`. In order to use the privacy extension, the kernel must be configured to support it. This can be done by enabling the configuration parameter `CONFIG_IPV6_PRIVACY` from the kernel configuration file, `.config`.

Enabling the privacy extension is not enough, network interface devices must also be configured to support it. Following sections discuss in more detail how the configuration can be done, and how global scope unicast addresses are generated for privacy extension enabled network devices.

3.1.1 Using the IPv6 Privacy Extension

Although the privacy extension has been implemented in Linux, applications cannot use it directly. Currently, there does not exist a common API that

would allow applications to choose whether to use public or temporary addresses. Usually applications use the default address selection procedure of the kernel, which prefers public IPv6 addresses over temporary ones. One default address selection procedure is defined by RFC 3484 [6] and is used, for instance, by the Linux kernel 2.6.15 [15].

Temporary addresses are generated and used only if the privacy extension is enabled in the kernel and network devices are configured to use them. Currently, configuration of the privacy extension can be done manually through the *proc* file system or *sysctl* interface (see Chapter 3.3). There exist also similar function interfaces for applications that can be used to configuration (the HIP privacy management uses these interfaces). Both of the interfaces, sysctl and proc, can be used to set values for privacy extension specific entries. The proc file system contains three such entries for each network interface. The entries are located in `/proc/sys/net/ipv6/conf/<interface>` directories. The following listing shows the privacy extension entries and explanations for the different values of them [2]:

- `use_tempaddr`
  - $<= 0$: disable privacy extension, only public address created.
  - $== 1$: enable privacy extension, public and temporary addresses created. Public addresses are preferred over temporary addresses.
  - > 1 : enable privacy extension, public and temporary addresses created. Temporary addresses are preferred over public addresses.
  - Default is 0 for most devices and -1 for point-to-point devices and loopback devices
- `temp_valid_lft` valid lifetime in seconds for temporary addresses. Default is 604800 seconds (7 days).
- `temp_prefered_lft` preferred lifetime in seconds for temporary addresses. Default is 86400 seconds (1 day).

For example, values can be assigned manually for `eth0` interface by using proc file system:
`echo 2 >/proc/sys/net/ipv6/conf/eth0/use_tempaddr` or sysctl interface:
`sysctl -w net.ipv6.conf.eth0.use_tempaddr=2`

There have been some suggestions how to implement an API extension that enables applications to override the default address selection rules. An Internet-Draft [24] describes an IPv6 socket API for address selection.

This address selection procedure is intended for hosts that use, for instance, the mobile IPv6, IPv6 privacy extension or cryptographically generated addresses. This socket API for address selection uses the basic IPv6 socket API [7] as a basis and simply adds a new socket option, `IPV6_ADDR_PREFERENCES` at the `IPPROTO_IPV6` level. The draft specifies several flags for the socket option, for instance, `IPV6_PREFER_SRC_TMP`, which indicates that the temporary IPv6 addresses must be preferred as a source address over the public ones. The draft specifies several different flags that enable different combinations of address types to be selected. Socket options are assigned with `setsockopt()` interface defined by RFC 3493 [7].

The USAGI project [28] has made a kernel patch, which represents another solution for temporary address selection. The idea is quite similar to the one that the address selection socket API describes above. It introduces also an additional socket option `IPV6_PRIVACY`, but the solution does not use flags. Instead, `setsockopt` receives simply a numeric value as a parameter (< 0, 0 or >0) that is used to define what kind of address selection actions should be performed when the `IPV6_PRIVACY` option is set to the IPv6 socket. The solution is purely for privacy extension purposes. Both techniques described above allow applications to choose per-connection based whether to use public or temporary addresses.

The HIP privacy management uses a different approach that will be described later. The socket APIs discussed above enable that different applications may choose different types of IPv6 addresses as a source address. This kind of behaviour may compromise the privacy of the host. The basic idea of the HIP privacy management is that it is hidden from the applications, and application cannot affect what addresses they use. It is not necessary to extend the IPv6 socket API in the context of this thesis because per-connection based address control is not what is needed. It is enough that the IPv6 privacy related system parameters can be altered by using the the proc or sysctl interface. By using these interfaces it is easy to control privacy parameters systemwide.

With the HIP privacy management a root user of the system can decide whether to protect all connection, or use the current, normal address selection scheme. Chapter 4 defines the requirements for the HIP privacy management prototype in more detail.

3.1.2 IPv6 Address Generation

IPv6 addresses are created and controlled in a single file inside the Linux kernel, `addrconf.c`. The kernel defines a couple of important structures for IPv6 devices and their addresses: `inet6_dev`, `ipv6_devconf` and `inet6_ifaddr` (see listings below).

The structure `inet6_dev` describes an IPv6 aware network device, `ipv6_devconf` configuration information for the network device, and `inet6_ifaddr` is structure for an IPv6 address of a network interface. The listings do not show all of the fields of the structures, only the ones that are important in this context.

```
struct inet6_dev
{
        struct net_device       *dev;
        struct inet6_ifaddr     *addr_list;
        ...
        ...
#ifdef CONFIG_IPV6_PRIVACY
        u8                      rndid[8];
        u8                      entropy[8];
        struct timer_list       regen_timer;
        struct inet6_ifaddr     *tempaddr_list;
        __u8                    work_eui64[8];
        __u8                    work_digest[16];
#endif
        ...
        struct ipv6_devconf     cnf;
        ...
}
```

```
static struct ipv6_devconf ipv6_devconf_dflt = {
       ...
#ifdef CONFIG_IPV6_PRIVACY
        .use_tempaddr           = 1,
        .temp_valid_lft         = TEMP_VALID_LIFETIME,
        .temp_prefered_lft      = TEMP_PREFERRED_LIFETIME,
        .regen_max_retry        = REGEN_MAX_RETRY,
        .max_desync_factor      = MAX_DESYNC_FACTOR,
        .privacy_mode           = IPV6_PUBLIC_MODE,
```

```
        .privacy_cycle          = TEMP_VALID_LIFETIME,
#endif
       ...
};

struct inet6_ifaddr
{
        struct in6_addr         addr;
        __u32                   prefix_len;
        __u32                   valid_lft;
        __u32                   prefered_lft;
        ...
        __u8                    flags;
        __u16                   scope;
        struct timer_list       timer;
        struct inet6_dev        *idev;
        struct rt6_info         *rt;
        struct inet6_ifaddr     *lst_next;
        struct inet6_ifaddr     *if_next;

#ifdef CONFIG_IPV6_PRIVACY
        struct inet6_ifaddr     *tmp_next;
        struct inet6_ifaddr     *ifpub;
        int                     regen_count;
#endif
        ...
}
```

IPv6 addresses are described by the structure `inet6_ifaddr`. Each IPv6 address has a valid lifetime and preferred lifetime. The lifetimes are determimed by the router advertised lifetime values or they are assigned the default values: one week and one day correspondingly. If the lifetime expires, the address is removed from the network device.

Each IPv6 address has a `flags` field that can be used to carry additional information about the address. For example, temporary IPv6 addresses have the flag `IFA_F_TEMPORARY` assigned. Each address has also a `scope` flag that tells whether the address is local or global.

If the privacy extension is enabled in the kernel, the fields `tmp_next`, `ifpub` and `regen_count` are included to the structure. The `tmp_next` and `ifpub` are temporary address specific fields. The first is a pointer to the next temporary

address of the network device, and the second is a pointer to the public IPv6 address that corresponds the temporary address. The public IPv6 address means in this context an address that is generated by using the stateless address autoconfiguration. The field `regen_count` is a counter used by the address generation algorithms.

When a network device is brought up first time, it must receive a Router Advertisement message that contains some global scope subnet prefix to be able to create a global scope IPv6 address. If the router is available and advertises the prefix, the public IPv6 address is generated by joining the prefix with the EUI-64 interface identifier. EUI-64 identifier is created from the link layer identifier if it is available (see Chapter 2.2.1).

If the privacy extension is enabled, a new temporary address is generated together with the new public address as defined in RFC 3041 [20]. The subnet prefix of the public address is copied and used as a subnet prefix of the temporary address. The field `rndid` of the corresponding network device, `inet6_dev`, has been initialized with random bytes. These bytes are used as an interface identifier part of the temporary address. The lifetime values are chosen to be the smaller value of the ones of corresponding public address or the ones of the network device configuration structure `ipv6_devconf`. The lifetime values listed in the structure `ipv6_devconf` can be assigned by the user with proc or sysctl interface. Lifetime values of a temporary address cannot ever be bigger than the values of the corresponding public address.

The source address selection is made based on the value of the `use_tempaddr` field of the `ipv6_devconf` structure. Each network device (`inet6_dev`) has its own configuration information (`ipv6_devconf`) stored in the field called `cnf`. If the value of `use_tempaddr` field of `cnf` structure is assigned to 2, it means that corresponding network device must use temporary addresses.

When the privacy extension is used, there must always be a valid temporary address available. The kernel checks periodically if the lifetime values of addresses are valid. When a temporary address deprecates a new temporary address is generated to replace the deprecated one. The kernel creates a new random value for `rndid` field of the network device and uses it as an interface identifier of the new temporary address.

3.2 HIP for Linux (HIPL)

The HIP privacy management is based on the HIP implementation, which was developed in the HIP for Linux (HIPL) project as a cooperation of

Helsinki Institute of Information Technology and Helsinki University of Technology. The implementation has evolved from the originally created HIP kernel module to a user space implementation. The HIP privacy management uses an intermediate form, which includes both HIP kernel module and user space HIP daemon. Most of the implementation issues of HIP are out of the scope of this thesis. The focus will be on the methods of how applications will identify their communication context using the HIP protocol.

3.2.1 HIP Socket APIs

All networking applications use sockets that denote communication associations between endpoints. Before any information can start to flow, both initiator and receiver side applications must create socket of a specific type, depending on the protocol they want to talk (for instance TCP or UDP) and associate their IP address and port with the socket. That is, two sockets are needed to describe the whole communication association [16].

Applications can use sockets with the help of different application programming interfaces (APIs) that define special functions for sockets. This section discusses the HIP socket API [16] and its differences compared to the current IPv6 socket API [7].

The HIP socket API is based on the IPv6 socket API [7]. It presents some new structures and modifies some of the existing API functions. There exists two different APIs for the HIP protocol: the *native* API for HIP-aware applications and the *legacy* API to be used with non-HIP applications. The legacy API enables applications to use HIP transparently without any modifications to the application code.

New Namespace Model

HIP introduces the Host Identity namespace between the transport and network layers of the TCP/IP stack. Figure 3.1 shows this new namespace model with the HIP protocol included. Each layer of the stack has its own identifiers.

The user interface layer uses a common identifier called Fully Qualified Domain Name (FQDN), which is a user friendly name referring to a network entity. The Endpoint Descriptor (ED) is one of the main ideas of the native HIP API. The ED serves as a pointer to the corresponding Host Identifier (HI) entry in the HI database of the host. The application layer uses EDs

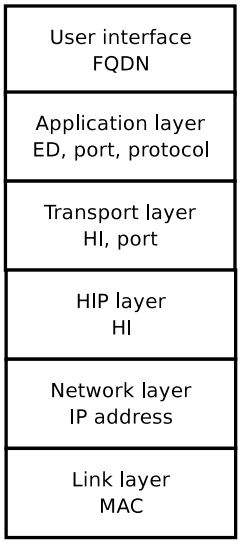

Figure 3.1: New namespace model and identifiers associated with each layer [16].

with sockets. The HIP sockets are associated with one source and one destination ED. Transport layer connections are associated with the source HI, destination HI, source port and destination port instead of IPv6 addresses and ports. Communication contexts in the HIP layer are bound to the source and destination HIs. The network layer uses IPv6 addresses that can now work purely as locators. The link layer is associated with hardware addresses. [16]

Resolver

The resolver is the part of the HIP API that is common for both native and legacy APIs. It provides a name and address binding service to the applications. From the view point of the HIP privacy management, resolver functions are one of the most interesting part of the API because applications can request source and destination HIs through them. Especially the resolver function `getendpointinfo`, which handles the hostname-to-address translations, is explored in more detail.

A HIP socket must be associated with the ED pointing to the local HI before information can start to flow. However, there exist different methods of how applications may initiate their sockets. These different methods are presented with the help of an example. Figure 3.2 illustrates a situation where a client

and server applications initiate a communication using the `TCP` protocol. The figure points the differences in resolver operations between the IPv6 socket API and the native HIP API. The HIP API does not change the interface syntax of other relevant socket API functions such as `bind`, `listen`, `connect` and `accept`.

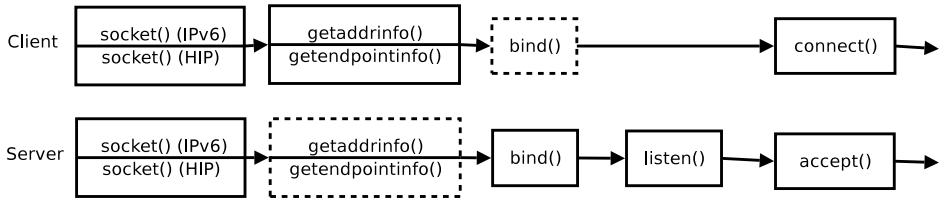

Figure 3.2: TCP connection using current socket API (getaddrinfo) or HIP socket API (getendpointinfo) [16].

First, the server creates a HIP socket. The server can optionally call a function called `getendpointinfo` if it wants to bind the socket to a specific source HI. That is, it allows connections request that contain a specific destination HI. If the server does not call `getendpointinfo`, it accepts connections to all of its HIs. Next, the server calls `bind`, which assigns the ED to the socket. Function `listen` indicates that the server is willing to accept connection requests. Function `accept` blocks until the connection is established with the client.

The client side creates also a fresh HIP socket. It queries the server's HI by using the resolver function `getendpointinfo`. The `bind` is not necessary at this point, because the `connect` call automatically assigns a suitable HI and port for the client to be used with ED. The bind function enables that networking applications can choose themselves which source HIs to use to communication.

The table 3.1 summarizes the different methods of how HIP networking applications may assign the source HI for their sockets.

3.3 Linux proc File System and sysctl

The Linux facilities proc and sysctl are used in the implementation of HIP privacy management. This section includes a brief overview of the usage and purpose of these facilities.

The proc is a virtual file system that exists only in the memory, and it is

Methods for assign source HI	Explanation
autobind	Autobind is performed by `connect`, application lets the system decide the source HI.
`getendpointinfo` + `bind`	HI queried from the local HI database and bound to the socket. Application can affect which type of HI is queried.
`bind`	Source HI is assigned by the application.

Table 3.1: Methods that applications can use to assign source HI.

created by the kernel during a system startup. The purpose of the proc is to allow user to read and configure the running kernel. The parameters in the proc file system are organized into a normal file system, and user can traverse directory tree and read parameter values from the files. Most of the parameters of the proc are read-only, but some parameters can also be modified.

Read-writable parameters are contained in `/proc/sys/` directory. This directory includes, for instance, variables that can be used to configure the IPv6 protocol (`/proc/sys/net/ipv6`). It will be discussed later how the HIP privacy management exploits these IPv6 specific parameters.

The parameters in `/proc/sys/` can be modified by using two different methods. The parameters may have so called proc names and they can be accessed through the proc file system, similar with other proc file system parameters. The other option is to use sysctl interface (see example 3.1.1).

Actually, the `/proc/sys/` part of the proc file system is usually called sysctl. The sysctl indicates also a user space application, which allows users to read and modify parameters through a command line. There exists also separate API, which allows other applications to read and write, or even create sysctl entries. [1]

3.4 Daemons in Linux

Another relevant and useful facility that needs to be introduced, are the daemon processes of Linux. A daemon is a process that runs autonomously in the background with or without any user interaction [30]. In order to implement a Linux daemon, one must follow a specific set of rules. The

following rules make the application an independent daemon process [30]:

- Separate the daemon from the parent process `fork`
- Change a file mode mask `umask`
- Open a log file
- Create a session id `setsid`
- Change a working directory `chdir`
- Close file descriptors `close`

The daemon process is moved to the background and separated from the parent process by forking. The command `fork` creates a child process that has a different Process ID (PID) and Parent Process ID (PPID) than the process that created it. The daemon inherits all the features of its parent except file locks and pending signals that are waited to be processed. The daemon inherits also file mode mask information that determines the file permissions.

The file mode mask is usually set to zero to prevent problems in file creation within the daemon. Setting the umask zero enables that user can properly read and write files created by the daemon.

Opening a log file is not mandatory, but it is very useful. Sometimes the log file may be the only place where to write debugging information.

The parent-child relationships are not enough to describe running processes under the Linux system. Instead, Linux uses groupings such as processes, process groups and sessions. A session can contain multiple process groups, which in turn can contain multiple processes. Process groups bind together processes that are working together. Sessions can be used, for example, to control which processes are run by which user. [14]

The daemon must change its process group during the initialization. The processes that have the same process group as the terminal, are in the foreground and are allowed to read from the terminal. When calling the `setsid`, a new session and process group is created and a new unique Session Identifier (SID) is returned to the daemon. New session must be created to be able to prevent daemon receiving unwanted signals from the shell. [8]

It is reasonable to change the working directory of the daemon as a root or other file system that contains all relevant information that the daemon

needs. If the daemon is started from a mounted file system, the file system cannot be unmounted before the daemon is killed. [8]

The daemon must close all open file descriptors that it has inherited from the parent process. Since the daemon cannot use the terminal, the open file descriptors are redundand and impose a security threat [30]. The function `getdtablesize` returns a maximum number of open file descriptors that the process can have. With that information all file descriptors can be closed.

The body of the daemon is simply an infinite while loop that performs wanted tasks. Usually it is required that there is only one instance of the daemon running at the time. Running multiple instances may have unexpected consequences. The daemon creates a lock file `/var/run/deamonnamed.pid` where it stores the process id (PID) of the running daemon instance. Everytime the daemon is started, it is checked if the lock file already exists. The daemon creation is stopped if the lock is reserved.

Usually daemons do not interact at all or very little with the user. One way to implement a interactive daemon is to use the Linux signaling facility. There are several standard signals in the Linux (ALRM, HUP, TERM, USR1, USR2 etc.), that can be used to signal commands to daemons. The interface that is used to send the signals is `kill`. It can be used from the command line or from code. The default behavior of the daemons, when the signal arrives, is termination. However, daemons can have separate signal handlers that are triggered when signals arrive to the daemons. Usually the signal handlers perform very minimal tasks. For instance, they may only change some boolean parameter, which causes the while loop to enter some `if` branch.

3.5 Summary

This chapter summarizes the Linux specific techniques used in the HIP privacy management implementation. The emphasis is on the IPv6 addressing scheme and the techniques how applications can choose their source HIs by using the HIP socket API. The chapter presented also the Linux sysctl facility and daemon creation process within the Linux operating system.

Chapter 4

Requirements

This chapter sets functional and non-functional requirements for the HIP privacy management prototype. The main focus is on functional requirements. The non-functional requirements concentrate mainly on how to facilitate the further devolopment of the prototype.

4.1 Functional Requirements

The HIP privacy management is a Linux program which purpose is to allow a HIP-aware host to secure its true identity and location when communicating across public networks by using the HIP protocol. More specific, users should be able to control the use of identifiers in three different networking layers: HIP layer, network layer and link layer. Networking applications that use sockets (IPv4, IPv6 or HIP sockets) have traditionally had a control over the identifiers they use to communication. For instance, they can bind their IPv6 sockets to a certain source IPv6 address, or HIP sockets to a certain source HI by using the services provided by socket APIs. Alternatively, they can let the kernel decide which identifier is the most suitable for their use.

The HIP privacy management removes the control from networking applications to the user. By using the HIP privacy management, the user can forbid applications to choose their own source identifiers. The HIP privacy management affects only HIP-aware applications that use HIP sockets to communicate on top of the IPv6 protocol.

A user can control the usage of identifiers by choosing between two different functional modes *stealth* mode and *normal* mode. The user can control the system only in a limited way. He cannot decide the source identifiers

by himself. Instead, he can only choose the mode, which gives the kernel and native HIP socket API rules of what kind of identifiers networking application are allowed to use. In addition to the source HIs, the HIP privacy management controls also two lower level identifiers, network layer identifiers (IPv6 addresses) and link layer identifiers (MAC addresses).

4.1.1 Modes of Operation

These two different modes, stealth and normal, correspond two different categories of identifiers: private identifiers in all levels and public identifiers in all or some levels. By using the HIP privacy management, a user can choose in which mode he wants the system to operate. It is very important that all applications, despite of which network interface they use, follow the rules of the current operational mode. Otherwise, the privacy of the host can be compromised.

Stealth Mode

The stealth mode means that all networking applications must be forced to use private identifiers in the following three layers: link layer, network layer and HIP layer. If the stealth mode is activated, the host can only initiate HIP connections.

The link layer privacy management is limited to support only Ethernet technology according to the IEEE 802 standard [26]. In the link layer, the use of a private identifier means that the default MAC address, provided by the network device manufacturer, must be overrided with a randomly generated MAC address. Moreover, the new MAC address must be marked as locally administered unicast address according to the IEEE standard.

If the host is multi-homed, the change operation must be done to all of its network interfaces. The reason why MAC addresses must be changed is that those can be used to trace the network behaviour of communicating hosts. Afterall MAC addresses are included in every Ethernet frame send to the network.

In the network layer a private identifier means an IPv6 address that does not reveal the valid link layer identifier of a host. An IPv6 address, generated through the stateless address autoconfiguration, contains an EUI-64 identifier based on the host's MAC address. The threat this scenario imposes is that the original link layer address can be restored from the EUI-64 identifier with some simple binary operations. Therefore, the MAC based EUI-64

identifiers must be replaced with random bit sequences. The subnet portion of the address must naturally contain the valid subnet identifier.

In addition of randomizing the MAC addresses and IPv6 addresses, these two identifiers must be changed periodically and in synchronized way. Changing the identifiers makes it difficult to trace the current and past locations of the host. The prototype must allow user to decide what is the most suitable change period of the addresses. The most natural change time is probably when the host actually changes the topological location, or decides to become active after being silent for a while. The prototype is not required to recognize the most suitable moments for changing the identifiers of the host, but the user must estimate these moments himself. The identifiers can also be changed during the data flow. The user should be able to alter the change period during the stealth mode.

In the HIP layer, the private identifiers mean the anonymous HIs of the host. The stealth mode must force HIP-aware networking applications to use only these anonymous HIs to initiate HIP associations.

All open HIP associations must be closed when stealth mode is activated. The HIP protocol uses public HIs as a default identifiers, and it is most likely that the open associations are bound to these default HIs. The prototype must warn the user about termination of associations and let the user decide whether he wants to proceed.

Normal Mode

The normal mode is the default mode in which the system operates normally. It means that both public and anonymous HIs can be used. The applications are free to choose what type of HIs to use to initiate the communication. Random MAC addresses and IPv6 addresses are not generated either. The most usual case is that applications use the default identifiers of the network devices. The host can also work as a responder of the HIP connections.

4.2 Non-Functional Requirements

The main focus of the implementation is on functional requirements. Non-functional requirements such as quality and performance have a lower priority. Afterall, the application is a proof-of-concept protype. This section describes some basic non-functional requirements that make the prototype more clear to use and understand and easier to extend further in the future.

4.2.1 Usability

The prototype must have a simple user interface. The program is controlled through a command shell. The user should be able to invoke the program, for example, by using the following command format: `program -options`.

The prototype must communicate with the user during the mode change. For example, if the open HIP associations are to be terminated during the set up of the stealth mode, the user must be able to cancel or continue the mode setup. During the normal mode the user may want to restore the original, manufacturer defined MAC addresses without rebooting the computer. The prototype must store the original MAC values of the network devices and allow the user assign these values during the public mode. The user must be able to close the prototype gracefully. During the close process, the prototype should assign the original MAC addresses to the network devices.

The graphical user interface (GUI) is not necessary, but the prototype should be easy to extend to include this interface as well. There is already a GUI available for the HIP protocol, and it may be convenient to integrate the HIP privacy management with the GUI in the future.

4.2.2 Expandability and Modularity

Expandability of the prototype is quite important, afterall, in the context of this thesis, it is not an intention to implement the whole privacy support. It must be possible to complete the prototype in such a way that it will provide the complete privacy support in the future. Also, the technological limitations should be easy to remove in the future. For example, the prototype should also support other network technologies than Ethernet.

Modularity is closely related to expandability. Modularity of the application facilitates the further development of it. The protope should include proper interface, which separates the technology specific tools from the common tools and allows to include other technologies to the prototype as well. For instance, the Ethernet specific functionalities should be separated from the common network interface tools.

4.2.3 Compatibility

The prototype should not change the functionality of the Linux TCP/IP stack or function interfaces. The prototype should only affect the source

identifier selection during the stealth mode in the three levels of the stack. The normal mode should work as the system usually does without any privacy management. The HIP socket interfaces should not be changed. The HIP privacy management must be totally transparent to networking applications.

The user should be able to use the HIP privacy management together with other networking facilities. For instance, the IPv6 privacy extension for the stateless address autoconfiguration should work normally with different modes of the HIP privacy management.

4.3 Summary

The purpose of the HIP privacy management prototype is to provide a HIP-aware host ability to secure its true identity and location when communicating across public networks by using the HIP protocol. The prototype allows user to choose between two functional modes: stealth mode and normal mode. The stealth mode forces HIP-aware networking applications to use private identifiers: anonymous HIs, random IPv6 addresses and random MAC addresses. The normal mode corresponds the current networking behaviour. The main focus of the prototype is on functional requirements.

The most important non-functional requirement is expandability, which means that the further development of the prototype should be made as easy as possible. Another important non-functional requirement is compatibility, which indicates that the prototype should not disturb other functionalities of the Linux operating system. Also, the prototype should not change the underlying function interfaces. The whole privacy management should be totally transparent to networking applications.

Chapter 5

Design

The HIP privacy management is neither an ordinary Linux user space nor kernel space application. It is a little bit of both. The privacy management affects three networking layers and allows user to control how the identifiers of these layers are used. Figure 5.1 presents the architecture of the HIP privacy management. The three layers in the figure present a conceptual model of layered structure of the networking implementation and identifiers associated with each layer. Each layer identifier is controlled by a corresponding layer module.

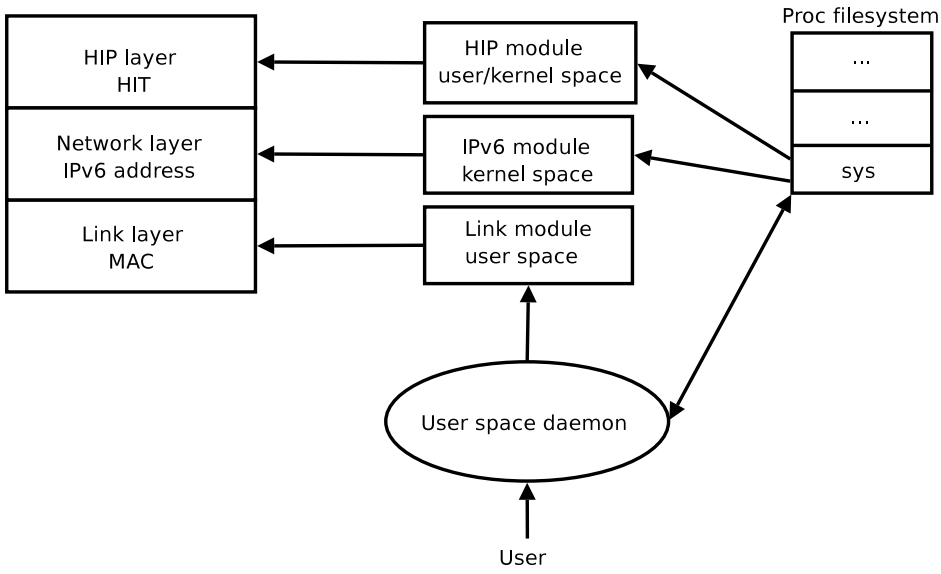

Figure 5.1: The architecture of the HIP privacy management.

The main part of the implementation is a user space daemon, which activates the privacy mode and sets values for privacy related entries in the proc file system (see Chapter 3.3). The HIP privacy management presents two new runtime, privacy related parameters: privacy mode and address change cycle. The proc file system communicates the values of these parameters between the user space and kernel space.

The link module controls the random MAC address generation. It contains also tools for controlling the network devices. This module is used by the user space daemon, which tells the module to periodically create and assign new MAC addresses. The word module refers to the parts of the implementation that are responsible for certain common tasks.

IPv6 addresses are controlled by the IPv6 module that lies in the kernel space. The user does not communicate with this module directly. The IPv6 module receives information about the values of the privacy parameters from the proc file system and changes its behaviour accordingly.

The user does not communicate directly with the HIP module. Same way as the IPv6 module, the HIP module receives the privacy mode information from the proc. When a networking application initiates the communication with stealth mode activated, the HIP module allows the application to use only anonymous HIs as source identifiers.

As an illustration, when user switches to the stealth mode, the daemon is activated with two parameters: privacy mode and address change cycle. The daemon stores these values to the proc file system. The daemon tells link module to generate and assign new MAC addresses for all network devices. Next, through a chain reaction, network module generates new randomized IPv6 addresses for the network devices. When the user initiates a new HIP connection, the HIP and IPv6 modules receive the information about the used mode from the proc file system, and assign source HIs and source IPv6 addresses according to the mode information.

The following sections discuss the design of the prototype layer by layer.

5.1 Link Layer

Figure 5.2 represents the privacy management design for the link layer. As mentioned above, MAC addresses are controlled by the user space daemon and the link module. The daemon calls periodically the link layer management module, which in turn tells the hardware specific tools to generate new

link layer identifiers for all network devices of the host. The hardware specific tools gets the random numbers for MAC addresses from the random generator of the link module. After the new MAC addresses have been assigned, the link layer management returns to wait the next call from the daemon.

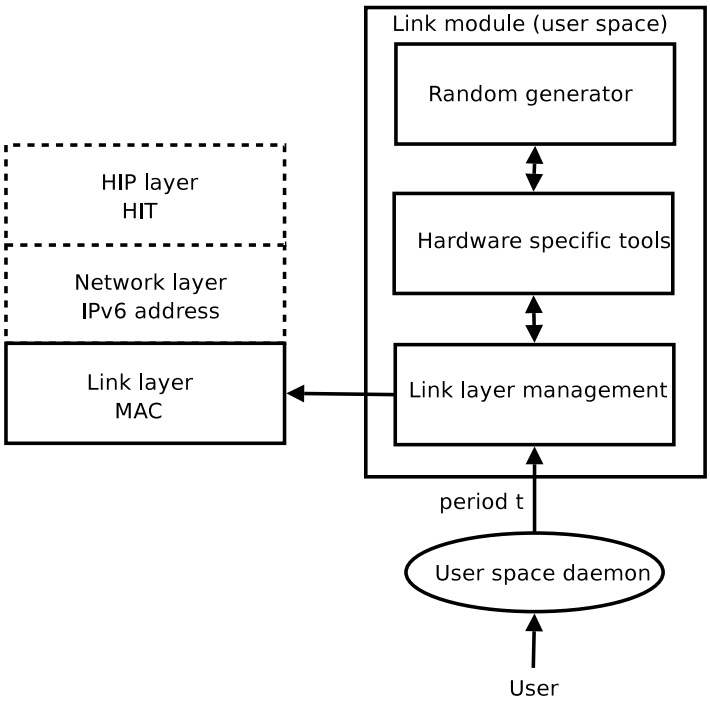

Figure 5.2: The architecture of the link layer privacy management.

5.2 Network Layer

Figure 5.3 shows the privacy management design for the network layer. The implementation is in the kernel space, inside the IPv6 protocol implementation. The network layer management listens changes of the sysctl parameters and updates its internal the privacy mode information accordingly. If the stealth mode is activated, the network layer management receives information about changed MAC addresses and generates new randomized IPv6 addresses for the network devices. When upper layers request source IPv6 addresses, the module assigns proper source addresses according to the current privacy mode. If the privacy mode is stealth, the IPv6 module allows the outgoing traffic to use only temporary IPv6 addresses.

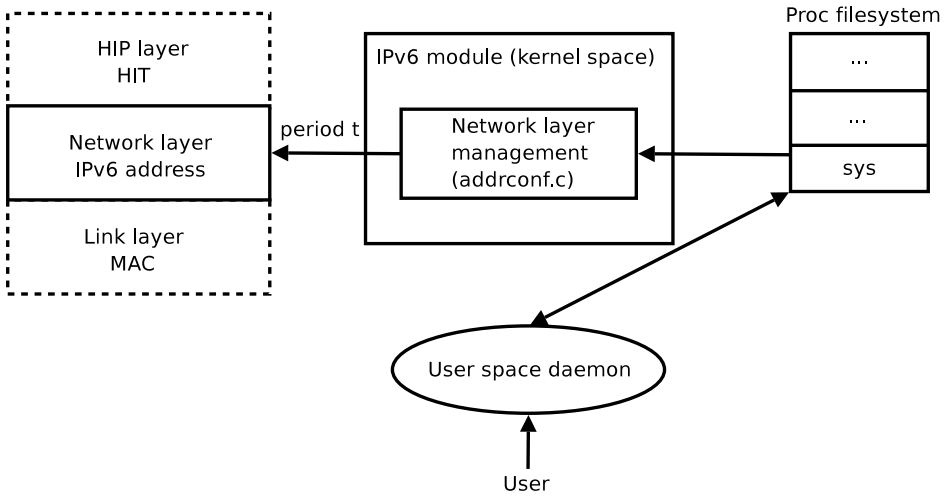

Figure 5.3: The architecture of the network layer privacy management.

5.3 HIP Layer

It can be seen from figure 5.4 that the principles of the privacy management in the HIP layer are similar to those in the network layer. The user does not communicate the privacy parameters directly to the HIP module. The information about changes comes from the sysctl. The module lies in the kernel space and user space. The reason for this is that the underlying HIP implementation is currently devided between the user and kernel space. Depending of the privacy mode, the HIP management module determines which HI is suitable for initiating a HIP association. The module controls also incoming HIP traffic and blocks the HIP connection requests if the stealth mode is activated.

5.4 Design Alternatives

The HIP privacy management application is a quite unusual application because it requires implementing a common control module that has access to different places of the Linux TCP/IP stack.

The reason why part of the HIP privacy management is in the Linux kernel is quite straightforward. The Linux TCP/IP stack is also inside the kernel. The prototype controls the layer identifier information and it is easiest to access the information in the kernel. Moreover, the kernel includes some useful

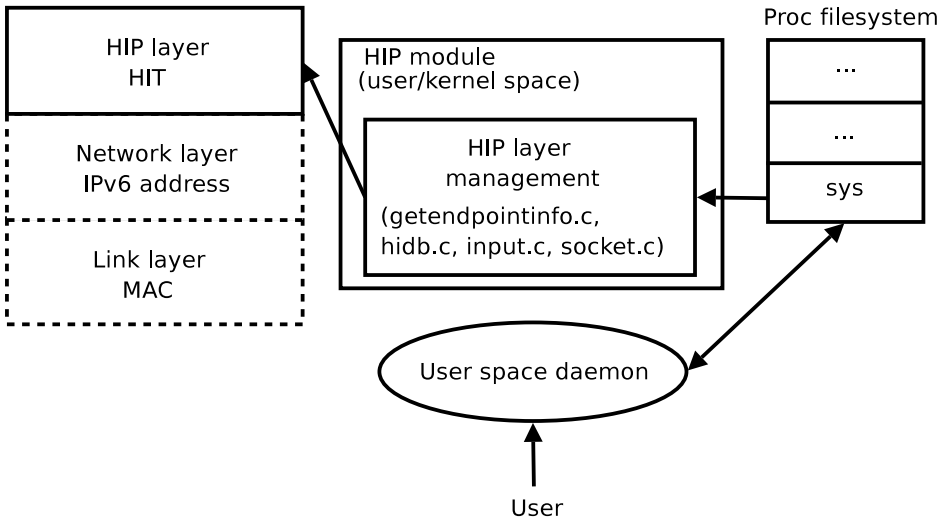

Figure 5.4: The architecture of the HIP layer privacy management.

features that are discussed later in this thesis. The HIP privacy management can exploit these features. For instance, implementing the HIP privacy management totally in the user space, would have required an implementation of a random generator for IPv6 addresses. The Linux kernel includes a random generator for IPv6 addresses; therefore, it was not practical to implement another random generator in the user space. If the whole HIP privacy had been implemented in the user space, it would have been more complicated. It would have required accessing such information that is available in the kernel.

5.5 Summary

The design of the HIP privacy management is divided between the user space and kernel space. The design consists of four different parts: user space daemon, link module, network module and HIP module. The user space daemon is the main module of the prototype. It communicates with the user and triggers events related to the certain privacy mode. The three layer modules control the usage of corresponding layer identifiers. The reason why the HIP privacy management is partly in the kernel is that it exploits the existing features of the IPv6 and HIP protocols.

Chapter 6

Implementation

This chapter discusses in detail the implementation of the HIP privacy management prototype.

6.1 User Space Daemon

The user space daemon (see figure 5.1) called `privacyd`, is the main module of the prototype. It handles the user interface, parses command line arguments, mediates the privacy mode and address change period parameters to the proc file system and controls the link layer management module.

The HIP privacy management is used as any other Linux program: `program -options`. For instance, the stealth mode is actived in the following way: `privacyd -stealth 10`.

The command causes the daemon to activate stealth mode with the change period of MAC and IPv6 addresses set to ten minutes.

6.1.1 The Implementation of the Daemon

The Linux daemon specific functionalities are implemented as described in Chapter 3.4. The user can interact with the running daemon simply by calling the program name with some options. For instance, if the daemon runs the stealth mode with the change period set to ten minutes, the command `privacyd -stealth 15` would change the period to fifteen minutes, or `privacyd -public` would change the mode to the normal.

The implementation uses the Linux signaling scheme to inform the daemon

that the user has given some new options for it. For example, user can set options that tell the daemon to change the mode or period, restore the original MAC addresses or terminate the program. The prototype uses three Linux signals for these purposes: USR1, USR2 and TERM. The daemon contains specific signal handlers for these three signals. The other Linux signals have been disabled to prevent an unexpected behaviour of the privacy daemon.

6.1.2 The Big Loop

Daemons usually have an infinite while loop that contains the main functionalities of the program. After the privacyd daemon has been initiated, it enters its while loop (see code in Appendix A).

In every iteration round daemon checks if the user has changed something or wants to terminate the program. If the current mode is stealth, daemon changes periodically MAC addresses of the host's network devices and sleeps between address changes. If the mode is normal, the daemon calls `pause`, which causes the daemon to sleep until some new signal arrives.

6.2 Link Layer

The link module (see figure 5.1) controls the link layer privacy management. The module lies in the user space, and it contains three distinct parts: link layer management, hardware specific tools and random generator (see figure 5.2).

When the user decides to switch to the stealth mode, the daemon stores the original MACs of the network devices into the linked list. Next, the daemon tells the link layer management to query new MACs from the hardware specific tools and assign them to the devices. This is done for all active network devices except for the local loopback, `lo`, device.

6.2.1 Link Layer Management

The link layer management contains general tools for handling network devices. These tools are used by the daemon, which does not know the details of the underlying network technology. The link layer management consists generic tools for MAC address generation and assignment. It contains func-

tions for checking the status of network devices and bringing the interfaces up and down. The link layer management uses the Linux `ioctl` commands to control network devices. There are also functions that can be used to query the harware types of the network devices.

6.2.2 Hardware Specific Tools

The hardware specific tools contains functions that can be used to generate hardware addresses for different types of network devices. Currently, this part of the link module consists only Ethernet specific functions. The hardware specific tools uses the structure called `hwtype` (see listing below) to define a generic function interface for network devices. Each network device type has its own `hwtype` structure and own implementations for the functions of the structure. The daemon can use these functions through the generic interface. The implementation of the hardware type support is based on the Net-tools 1.60 software package [5].

```
struct hwtype {
    char *name;
    char *title;
    int type;
    int alen;
    char *(*print) (unsigned char *);
    int (*check) (char *, struct sockaddr *, int);
    struct sockaddr *(*new_mac)();
}
```

6.2.3 Random Generator

The random number generator is used by the harware specific tools to generate random link layer identifiers. The random generator is based on the implementation in a Secure Programming Cookbook for C and C++ [29]. The book represents a common API for randomness and entropy. According to the book there exist three classes of solutions of random generators:

- Insecure random number generator
- Cryptographic pseudo-random number generator
- Entropy harvester

The first type is not very recommended to use, because the generated numbers may be forseeable. The solution used here is a combination of the second and third type of generator. The generator returns entropy if it is available, and otherwise it returns cryptographically strong random numbers by using any available entropy.

The implementation uses two sources of randomness that most Linux operating systems have: `/dev/random` and `/dev/urandom`. The first one generates data that is statistically close to pure entropy. The second provides only cryptographic randomness. However, it must be noted here that the quality of entropy depends on the implementation of the operating system and may vary between different Linux systems. [29]

6.2.4 Address Generation

An Ethernet address can be one of the following types:

- A physical address of one network device (unicast address)
- A network broadcast address
- A multicast address

These different formats of Ethernet addresses must be taken into account while random MACs are generated. The goal is to generate only locally administered unicast hardware addresses. The harware specific tools requests six random numbers from the generator. Those numbers are converted to hexadecimanl format, concatenated together and sended to the checking.

To make sure that the address is in right format, the group bit is turned off and local bit is turned on by the address checking function of the harware specific tools. Basically this means that the generated MAC will be random, but the second value of the first byte can have four possible values instead of sixteen. The third row of the table 6.1 illustrates the values that the second value of the first byte can have.

6.3 Network Layer

The IPv6 module (see figure 5.1) lies in the Linux kernel. The whole IPv6 module is part of the Linux IPv6 implementation. The implementation can

0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	**0010**	**6**	**0110**	**A**	**1010**	**E**	**1110**
3	0011	7	0111	B	1011	F	1111

Table 6.1: Hexadecimal numbers and the corresponding binary values.

be found from the `/net/ipv6/` directory of the kernel tree and is based on the 2.6 kernel development series.

The network layer privacy management exploits the privacy extension for stateless address autoconfiguration (see Chapter 2.2). Why this feature is used is that it provides a random generator for IPv6 addresses and ways to assign these addresses to the network devices. Actually the word random is a little bit misleading here because only the host portion of the IPv6 address is generated randomly, network portion must naturally be a known subnet address.

It must be noted here that to be able to use the HIP privacy management, the Linux kernel must be configured to support the IPv6 privacy extension. This can be done by enabling the configuration parameter `CONFIG_IPV6_PRIVACY` from the kernel configuration file. The sysctl facility must also be supported, that is, the parameter `CONFIG_SYSCTL` must be enabled.

6.3.1 Network Layer Management

Figure 6.1 represents the part of the HIPL source tree, which is essential from the view point of the HIP privacy management. The figure also shows the context of the files, that is, whether they are located in the user space or kernel space.

The network layer management is located in the file `addrconf.c` and the corresponding header file `addrconf.h`. The header file defines two new privacy related parameters `IPV6_STEALTH_MODE` and `IPV6_PUBLIC_MODE`. The values for these parameters are 2 and 1 respectively. It can be noted here that these values follow the privacy extension rules discussed in Chapter 3.1.1.

The sysctl

The sysctl plays an important role by communicating privacy related information between the user space and kernel space. The sysctl contains two new

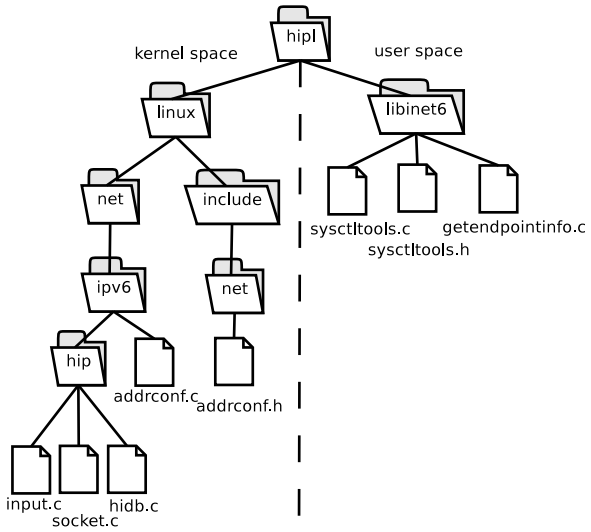

Figure 6.1: Implementation of the network layer and HIP layer privacy management.

privacy related entries for each IPv6 enabled network device: `privacy_mode` and `privacy_cycle`. The implementation of these new entries is included in the file `addrconf.c`. The implementation of the HIP privacy management includes a user space tool for quering the sysctl entries: `sysctltools.c`.

The sysctl calls are handled by the specific handler functions in the kernel. These functions are called when the kernel perceives that the sysctl call is invoked. The kernel includes three handler functions for both privacy entries. For instance, `privacy_mode` entry has one handler for the proc interface, which is called when the user enters `privacy_mode` parameter by using the proc interface. The other handler is for the sysctl interface, and it is used to read and write the `privacy_mode` entry. The third function is used to propagate the privacy information further inside `addrconf.c`. The `privacy_cycle` entry has also corresponding three handlers.

Currently, only the `privacy_mode` entry is used in the actual network layer privacy implementation. The `privacy_cycle` entry is included for informative purposes for a user to be able to view the address change period similar manner with other runtime system parameters.

Eventhough it is possible to modify these two sysctl entries manually through a shell (or some other application) without using the actual HIP privacy management, it may cause the system to behave an undesirable way.

Address Generation

The most important change for the HIP privacy management is included in the function `ipv6_create_temp_addr` in the `addrconf.c` file. This function generates a new temporary address for certain public IPv6 address. It copies the valid network portion of the public address and merges it to the randomly generated host portion. The function assigns suitable lifetime values for newly generated temporary address based on the lifetime values of the public address.

The IPv6 address change is synchorinized with the MAC address change. The synchronization is implemented by exploiting the fact that the network interface device must be brought down before a new MAC can be assigned to it.

When a network interface device is brought down, the IPv6 addresses of the device are deleted. However, when the network interface device is brought up, it can be seen that the functions in `addrconf.c` generate exactly similar IPv6 addresses than the device had before. Even the temporary IPv6 address is similar. This phenomenon can be explained by a field called `rndid` in `inet6_dev` structure presented in Chapter 3.1.2. This field includes the random host portion of a temporary IPv6 address, and it is reused when the network device is brought up.

The HIP privacy management changes slightly the temporary address generation. When the stealth mode is activated, and a network device is brought up with a new random MAC address, new public IPv6 address is generated normally according to the rules of the stateless address autoconfiguration. The difference is that when the corresponding temporary address is generated, the system is forced to generate a new random value to the field `rndid` of the network device.

Source Address Assignment

In addition to the IPv6 address generation, another important issue is the IPv6 source address assignment. Functions `ipv6_dev_get_saddr` and `ipv6_get_saddr` are used to assign the most suitable source addresses to the network devices. The actual source address selection functions are not changed, but it is made sure that the functions always return temporary addresses when stealth mode is activated.

The source address selection for a specific network device is made based on the `use_tempaddr` field of the network device's configuration structure

`ipv6_devconf` (see Chapter 3.1.2). When `privacyd` daemon updates the `privacy_mode` entry in the sysctl, the specific sysctl handler assigns suitable value for the `use_tempaddr` field of each network device. If the mode is stealth, the values of the `use_tempaddr` fields of all network devices must be set to 2. This value indicates that the network devices must use temporary addresses (see Chapter 3.1.1).

6.4 HIP Layer

The HIP module (see figure 5.1) lies in the user space and kernel space. The native HIP socket API has a user interface (for example `getendpointinfo.c`) in the user space and socket handler functions in the kernel space (`socket.c`). Figure 6.1 shows the file hierarchy in more detail. The HIP privacy management required also some changes to the following kernel components of HIP: `input.c` and `hidb.c`.

The HIP layer privacy management forces HIP-aware applications to use only anonymoust HIs as a source HI. The implementation needs to take into account the following four cases (see table 3.1):

- Host uses autobind to get source HI (`hidb.c`)
- Host uses `getendpointinfo` to query a source HI (`getendpointinfo.c`)
- Host uses `bind` to set manually some source HI (`socket.c`)
- Host receives a HIP connection request (`input.c`)

6.4.1 autobind Case

The autobind case is handled in the file `hidb.c`. The HIP daemon has a database where it stores the local and peer HIs. If some application queries local HIs from the database and the stealth mode is detected, only anonymous HIs are returned. The anonymity can be checked by using the `anon` field of the `hip_lhi` structure, which is used to represent local HIs:

```
struct hip_lhi{
    uint16_t           anonymous; /* Is this an anonymous HI */
    struct in6_addr    hit;
}
```

6.4.2 getendpointinfo Case

This case is handled in the user space in the file `getendpointinfo.c`. If the host decides to use the function `getendpointinfo` to query some specific local HI and the stealth mode is detected, the function is forced to return only anonymous HIs.

6.4.3 bind Case

The `bind` socket API function is in the kernel and required some changes to the HIP specific socket handler `socket.c`. The HIP privacy management is handled similar way than in the autobind case. The `bind` function has an access to the `hip_lhi` structure, which can be used to check the type of the selected HI.

6.4.4 Incoming Connection Requests

A HIP host that uses the stealth mode can only initiate connections. The connection requests are blocked in the file `input.c`. When the host receives the I1 message (see figure 2.4), it does not do anything.

The HIP Internet Draft [19] specifies that the HIP-aware host may choose not to accept a HIP base exchange and should send the Destination Unreachable message specified by the Internet Control Message Protocol (ICMP) to the peer (this feature is not implemented in the current HIP version). However, in the case of the HIP privacy management, host does not send anything to the peer because sending the ICMP message would reveal that the host is present and owns the specific HI.

6.5 Summary

The main module of the prototype is a Linux daemon. The user can interact with the daemon through signaling. The prototype uses the following Linux standard signals: USR1, USR2 and TERM. The daemon overrides the default signal behaviour with the specific signal handler functions.

The link layer privacy handling is done in the user space. The implementation includes random generator for MAC address generation and also Ethernet specific tools for controlling Ethernet network devices.

The network layer implementation exploits the existing Linux IPv6 implementation and especially the feature called the IPv6 privacy extension. This feature provides a way to generate and assign new temporary IPv6 addresses for network devices.

The HIP level privacy management is focused on the methods of how applications can choose the source HIs for their sockets. There exist three different methods that the HIP privacy management controls: `autobind`, `getendpointinfo` and `bind`.

Chapter 7

Analysis

This chapter concentrates on evaluating the implemented prototype against the requirements presented in Chapter 4. Although there were not any specific performance requirements for the prototype, this chapter presents some performance measurements of the HIP update mechanism. The update mechanism is closely related to MAC and IPv6 addresses changes during the stealth mode.

The prototype was evaluated by using both black-box and white-box testing methods. The difference between these methods is that the black-box testing does not utilize the knowledge about the internal structure of the program. Test cases are performed by giving some valid input values to the system, and results are analyzed based on the correctness of the output values. White-box testing, also called as structural testing, uses the internal structure of the program as a basis when test cases are designed.

7.1 Evaluation of the Prototype Against Functional Requirements

Two different approaches are used to test the functional requirements. One approach is to use system testing, which means that the tests are performed to the whole integrated system. The system testing belongs to the category of black-box testing. Another approach is to use white-box testing. For instance, the different if-branches in the daemon's while loop (see Appendix A) need to be tested separately.

7.1.1 Testing Tools and Environment

The HIP privacy management was tested in a virtual environment provided by VMware Workstation 5.0.0. VMware is a virtual machine software, which enables that several virtual operating systems can be run on a single computer. The workstation allows also that the virtual hosts can be configured to different types of virtual networks.

This testing environment consists three virtual machines, one IPv6 router and two HIP hosts that communicate with each other. All virtual hosts have 2.6.11. Linux kernels with HIP support. Another testing tool is Ethereal. Ethereal is a network protocol analyzer, which allows realtime network packet capturing and reading the content of the packets.

Figure 7.1 shows the topology of the test environment. The two virtual machines communicate by using HIP, and the real host listens the HIP traffic and analyzes it with Ethereal. The virtual HIP host 1 runs the HIP privacy management prototype. The IPv6 router advertises the global scope prefix of the virtual network and enables that the hosts can use stateless address autoconfiguration to generate their global scope IPv6 addresses. The virtual machines do not have an access to the public Internet.

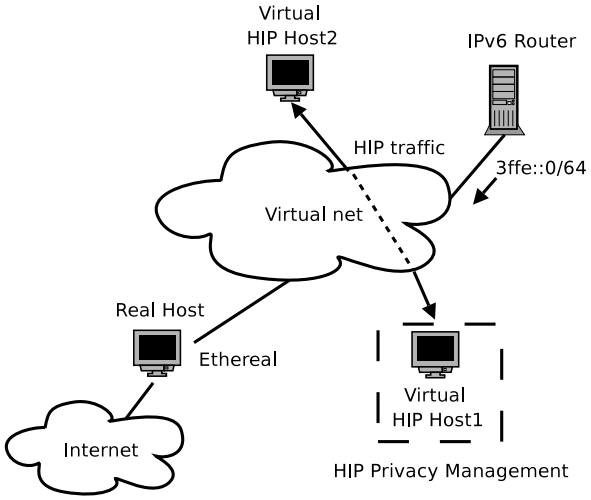

Figure 7.1: The virtual test environment.

Other programs used for testing were ifconfig and telnet. Section 7.1.2 presents also a test setup, where a specific server-client application is used for performance measurements.

7.1.2 Modes of Operation

The requirements stated that the HIP privacy management should be a Linux program that allows a HIP-aware host to secure its true identity and location when communicating across public networks by using the HIP protocol. The requirements defined two functional modes that user should be able to choose: stealth mode and normal mode. The stealth mode is a state where networking applications are forced to use random MAC addresses, temporary IPv6 addresses and anonymous HIs. Randomized identifiers protect the true identity of the host. The use of anonymous HIs causes that the host can only initiate HIP connections. Third parties cannot reach the host by using its public identifiers. The normal mode is the current state, where applications are free to choose their identifiers.

The prototype is not a traditional Linux program because the implementation consists of different modules that are distributed between the user space and kernel space. However, the main program is an independent Linux daemon that triggers privacy related events and listens user commands.

The functional tests concerned mainly the stealth mode, afterall, the normal mode does not add any new functionality to the system. During the normal mode the daemon just sleeps and waits signals from the user.

Stealth Mode

Currently HIP-aware applications can choose which HI they use to communication. The stealth mode forces HIP connection initiators to use randomized identifiers in the three layers such as the requirements stated. By manipulating the implementation part where the connection requests are handled, the host that uses the stealth mode does not respond to any new HIP base exchange requests. The host is invisible to the other parties. Of cource, the peers that have HIP associations with the host, know that the host is reachable and owns a certain HI.

The support for the Ethernet technology is enough for a proof-of-concept prototype. The generated MAC addresses follow the IEEE 802 standard [26] such as required. Two bits of the MAC addresses are manipulated in such way that the addresses become locally administered unicast addresses. In the case of a multi-homed host, MAC addresses are changed for all network interfaces.

The prototype stores the original MAC addresses of the devices, and the user can restore those values if he wants. However, the prototype does not have

an error handling for situations where the user has removed some network interface during the stealth mode and tries later to restore the original MAC address for that interface.

The implementation exploits the IPv6 privacy extension such as the requirements necessitated. The IPv6 addresses are generated by means of stateless address autoconfiguration and randomized by using the IPv6 privacy extension. That is, randomized IPv6 addresses are actually temporary addresses defined by RFC 3041 [20]. The HIP privacy management manipulates temporary address generation in such a way that it generates new interface identifier part every time the temporary address is generated. This ensures that temporary addresses are always different.

The requirements stated that the MAC and IPv6 addresses need to be changed periodically such as the IPv6 privacy extension changes the IPv6 addresses. The change is done in a synchronized way by exploiting the fact that the network interfaces must be brought down in order to change the MAC addresses. Bringing the interfaces up and down causes a chainreaction, which makes the kernel to create new temporary IPv6 addresses for the network interfaces. This kind of a solution was chosen because using some timers would have required more changes to the kernel and would have been more complicated solution.

The user can decide the address change cycle and also change the cycle during the stealth mode. This is not the most optimal thing because the most natural change times would be the times when the host actually changes his topological location, or decides to become active after being silent for a while.

System's behavior during the stealth mode was tested in several different ways. The prototype was implemented layer by layer, and solutions for distinct layers were tested separately in addition to the system tests. The testing was done mainly by observing the network traffic with Ethereal. The prototype prints lots of debug information, which makes it easy to follow the control flow of the program. When testing the kernel modifications and behaviour of the daemon, the debug information was sent to the system logs.

The underlying HIP protocol implementation, HIPL, sets some limitations to the implementation of the HIP privacy management prototype as well as to the testing. According to the requirements, initiation of the stealth mode should close all open HIP associations. However, the close operation of the HIPL was broken at the time of implementation, and it was decided that it is not a good idea to correct the operation from this specific version of the HIPL. It is quite straightforward to add this feature later to the HIP privacy management after the bug fix. Currently, when stealth mode is activated, the

prototype warns the user about possible open HIP associations and allows the user to cancel the stealth mode initialization. Any open HIP associations need to be closed manually with some other Linux tools (for example by using ip program).

Normal Mode

The normal mode does not bring any new functionality to the system. The networking applications are allowed to use any identifiers they want. The only thing that needs to be taken care of is that the daemon must sleep during the normal mode. Otherwise it would only consume processing time unnecessarily. Only functionality that can be performed during the normal mode is restoring the default MAC addresses to the network devices. This feature was tested with system tests and with the help of ifconfig program that allows viewing the network interface parameters.

Performance Measurements

One quite interesting feature of the HIP privacy management is that it changes the MAC and IPv6 addresses of the host periodically. Also, in order to do that the prototype will have to bring the network interfaces down. With the HIP protocol this is not a problem because the HIP associations are identified by the HIs instead of IPv6 addresses, and identifiers under the HIP layer are free to change. The update mechanism of the HIP ensures that the location (IPv6 addresses) of the hosts can change without any connection breaks.

It is not a usual scenario that the network interface is brought down while there are still active connections on that interface. Usually, this kind of a event occurs when some communication link breaks. According to the tests, the HIP association recovers well from the link breakage, but the host that runs the HIP privacy management, experiences quite considerable delay in its traffic.

This delay was measured by using a client-server application where the server side sends continuosly data through a statefull TCP connection. Figure 7.2 shows the test setup. Two hosts, one client and the other server, initiates a TCP connection by using the HIP protocol. The client runs the HIP privacy management prototype. The server side sends its own timestamps in every 0.1 seconds. When the client receives the timestamp, it prints the servers timestamp together with its own timestamp to the standart output.

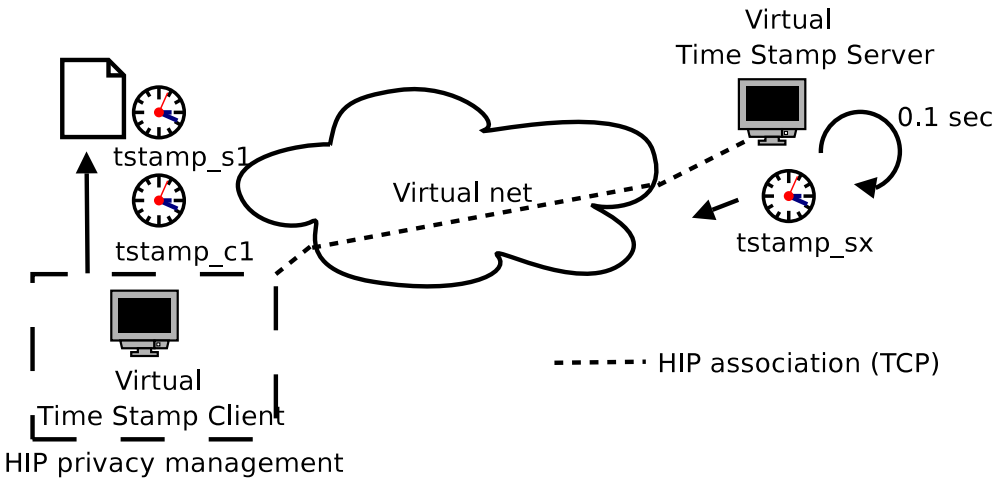

Figure 7.2: Test setup for delay measurements.

Two different measurements were performed. In the first case, the client uses the normal mode. The client changes its IPv6 address in every six minutes. That is, the client simulates a normal mobile host that uses the HIP update mechanism to inform the server about its location changes. During the second test, the client uses the stealth mode with address change period set to six minutes as well. The client uses the update mechanism also in this case, but the difference is that now both MAC and IPv6 addresses change. The MAC address changes cause also link breakages.

Figure 7.3 presents the measured delay in the case of the normal mode. The delay is measured between consecutive packets that the mobile client receives. The diagram shows a scene where the client receives approximately 20 000 timestamps. The vertical axis shows the time in seconds, and the horizontal axis shows the number of the received packets.

It can be seen from the figure that the delay between packets is only minimal. The packets are sent in every 0.1 seconds and the delay between consecutive packets is approximately between 0.05 and 0.25 seconds. The times under 0.1 seconds are explained by buffering. If the packets arrive to the host faster than it can process them, it stores them to the buffer to wait for processing. The diagram shows no specific delay pattern. It is impossible to say from the diagram when the address changes occur during the measurement.

Figure 7.4 represents a corresponding situation, but now the host uses the stealth mode. The graph shows delay measurement of 20 000 consecutive packets. It can be seen from the figure that link breakages cause quite no-

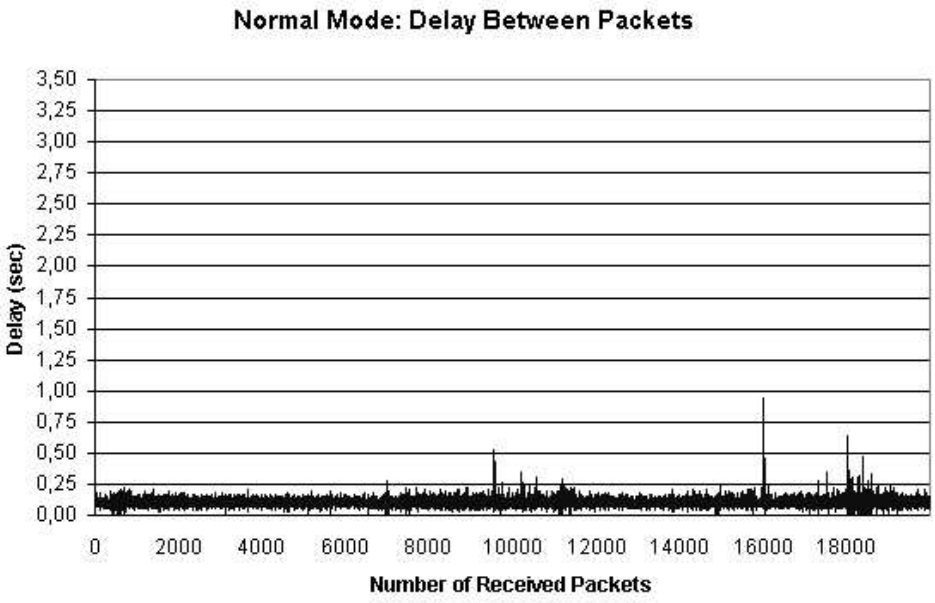

Figure 7.3: The delay between consecutive packets during the normal mode.

ticeable delay, between 3.00 -3.50 seconds.

To summarize, if a host uses the stealth mode with some networking application that uses statefull connection, the address changes can disturb the connection badly. Of course the user may try to optimize the address change period such way that it will not disturb his open connections.

It must be noted here that the results of the measurements are only suggestive because the test environment was virtual. Real hosts in the real network would probably perform much better with smaller delay times.

7.2 Evaluation of the Prototype Against Non-Functional Requirements

Non-functional requirements affected the design of the prototype. Evaluating the prototype against the non-functional requirements was performed by reviewing the design and code of the prototype and also with the help of the observations that were made during the functional testing.

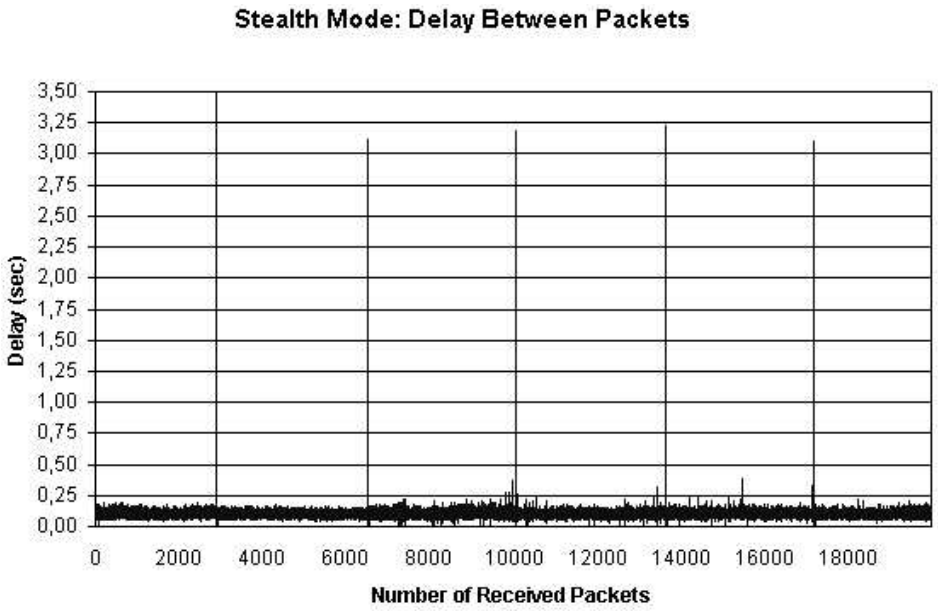

Figure 7.4: The delay between consecutive packets during the stealth mode.

7.2.1 Usability

The lack of the close operation degrades the usability of the prototype. It makes the prototype more complicated to use because the user has to remove the possible open HIP associations manually. Nevertheless, the user interface of the prototype is quite simple and similar to other Linux applications. The prototype can be controlled with only a few simple options.

It is also quite easy to extend the prototype in the future to include the application interface. This interface would make it possible to use the HIP privacy management through some external application. Especially the usability would improve if the HIP privacy management was integrated with the HIP GUI. The external applications will not need to access the data structures of the HIP privacy management, which makes the interface functions quite simple. That is, the external applications would only send signals to the HIP privacy management daemon.

7.2.2 Expandability and Modularity

The requirements stated that the prototype should be expandable. For example, it should be able to complete the privacy managament to support

full privacy, including anonymity and linkability, in the future. Moreover, support for other technologies than Ethernet should be easy to add later.

Extending the prototype to support the full privacy management depends mostly on the implementation of the HIP protocol, not the current HIP privacy management prototype. To extend the privacy, some protocol fields need to be protected and this must be done inside the HIP protocol implementation. It is rather difficult task to predict how the full privacy management support should be implemented. Afterall, the underlying HIPL implementation is not in its final format yet and also with the given time limits it was impossible to study the implementation of the HIPL further.

Currently, the HIP privacy management supports only Ethernet technology defined by the IEEE 802 standard [26]. However, the implementation of the prototype is designed to be modular, and it includes separate interface for the link layer functionalities. The interface hides technology specific details from the user space daemon.

7.2.3 Compatibility

The compatibility of the prototype was ensured with careful desing. The changes that were made in the kernel do not change the existing function interfaces. New functions that are added to the kernel are used strictly to the HIP privacy management purposes, and they do not affect other kernel functionalities. Also, changes to the existing kernel functions are very minimal, which makes it quite easy to test and control them.

7.3 General Analysis of the HIP Privacy Management

The HIP privacy management was a rather challenging program to design. It required a deep knowledge of the Linux TCP/IP stack and kernel. It was necessary to study the system layer by layer. This is the reason why the design and implementation proceeded parallel and iterative way. The design is probably not the most optimal for production purposes, but good enough for a prototype.

The requirements were met quite well, the only functionality that is actually missing is the closing of existing HIP associations. It is quite hard to estimate how well the non-functional requirements were met, but it can be stated that

they improved the architecture of the program.

Performance measurements were made in the virtual environment. The purpose of the measurements were to give an insight how the prototype would affect the connections in a real network environment. The delay of recovering from the MAC and IPv6 addresses changes is quite disruptive and would greatly disturb the normal usage of the applications that use statefull connections. One solution to round this problem would have been to use so called dummy device. The dummy device is a virtual device, which is not related to any physical network device. From the view point of applications, it is similar to any other network device and can be used to replace the real network device. In the HIP privacy management it could have been used to receive packets while the real network device is changing the MAC and IPv6 addresses.

One possible problem of the HIP privacy management is constantly changing identifiers. Firstly, not all network devices support MAC address changes, which may cause system limitations to the privacy management application. Secondly, the random generation of the MAC addresses is quite problematic; how one can be sure that the MAC address is random enough.

Third problem is address collisions that are handled by the IPv6 protocol that runs Duplicate Address Detection (DAD) for all unicast addresses. This method checks also MAC addresses, after all, uniqueness of addresses generated through the stateless address autoconfiguration is determined by the MAC based interface identifiers. However, accoring to the RFC 2462 the DAD is not totally reliable, and it is possible that all duplicate addresses are not detected [27].

Changing identifiers may also set environment limitations to the HIP privacy management. Not all networks support constantly changing identifiers. For instance, many services provided by access networks (for example company intranets) require knowledge about the IP addresses or MAC addresses of the hosts. Therefore, a host using the stealth mode could not be able to operate normally in this kind of networks. The HIP privacy management can anly be used in special type of network environments. One option would be, for instance, open access networks.

The purpose of the privacy management is to protect the privacy of the host, but at the same time it can reveal some information about the host. The level of privacy depends on the network where the host is located. For instance, if the network includes only a few hosts and amount of traffic is low, it is rather easy for eavesdroppers to notice that some host is constantly changing its identifiers. Another problem that makes it easy to guess the

usage of stealth mode is that the identifiers of the destination host do not change. By following the destination identifiers of the messages, it can be quessed that the source host is just changing its identifiers.

One solution for this problem would be the Onion Routing [4]. The idea of Onion Routing is to protect the privacy of the sender and receiver and also protect the content of the message. This task is accomplished by sending messages through a sequence of onion routers that re-route the message and also encrypt the message between each router. Re-routing makes it difficult to trace the path of the packets.

Chapter 8

Conclusions

The goal of this thesis was to design and implement a system, that enables HIP-aware hosts to communicate securely without revealing their true identity and location to any other network user. To be more specific, the goal was to solve partly these problems and provide a groundwork for more thorough HIP privacy support. The implemented prototype of the HIP privacy management solves the problems of pseudonymity and location privacy.

The basic idea of the HIP privacy management is to force all HIP-aware networking applications of the host to use private identifiers: random MAC addresses, temporary IPv6 addresses and anonymous HIs. The user can decide to use these private identifiers by choosing between two different operational modes: stealth mode and normal mode. The stealth mode is the state where all HIP applications must use these private identifiers and also change MAC and IPv6 addresses periodically. The stealth mode makes the host transparent in such way that it cannot respond to any new HIP connection request. The normal mode is the current situation where applications are free to choose their identifiers.

The implementation of the prototype is based on the Linux 2.6.11 kernel with the HIP support. The implementation is distributed into the user space and kernel space. The identifiers of the three layers are controlled in different ways: random MAC addresses are generated and controlled in the user space, temporary IPv6 address generation utilizes the IPv6 privacy extension and the control for anonymous HIs uses the existing HIPL implementation.

The prototype meets the set requirements well. Some compromises must be made because the underlying HIPL implementation is not in its final format yet. The design of the privacy management prototype is not the most optimal and requires further development and quality assurance to meet

the requirement of the production level software. For a proof-of-concept prototype the design and implementation are adequate.

The observations that were made during the process, were that the complete privacy management is very extensive area and needs lots of studying and experimenting the functionality of the current TCP/IP stack and the HIP protocol. The HIP privacy management works well inside the limited, virtual test environment. However, constantly changing identifiers may affect unexpected way to other functionalities of the real systems, or services of the real networks.

8.1 Future Work

Future work related to the HIP privacy management includes extending the privacy support. Basically this means solving the linkability problem by protecting the SPIs of the ESP traffic. This can be done in several different ways. For example, by negotiating new SPIs beforehand and udpating them within the update mechanism.

Currently the prototype changes MAC and IPv6 addresses of the host. This should be extended to include HIT changes as well. Changing the HITs periodically would extend the current pseudonymity support into the anonymity support.

Appendix A

Body of the HIP Privacy Management Daemon

```
/* The Big Loop */
 while (1) {

   // check if privacy parameters have been changed
   if(changed && !quit){
     if(get_mode(&mode) < 0 || get_cycle(&cycle) < 0){
       HIP_ERROR("big loop: get_mode or get_cycle failed\n");
       exit(EXIT_FAILURE);
     }
     changed = 0;
   }

   // check if original MACs can be restored
   if(original && !quit){
     int tmp_mode = 0;
     if(get_mode(&tmp_mode) < 0){
       HIP_ERROR("big loop: get_mode failed\n");
       exit(EXIT_FAILURE);
     }
     // mode is normal, restore original MACs
     if(tmp_mode == IPV6_PUBLIC_MODE && !default_ids){
       if(restore_default_mode() < 0){
         HIP_ERROR("big loop: restore_default_mode failed\n");
         exit(EXIT_FAILURE);
       }
```

```
        clean_link_list(list);
        default_ids = 1;
        first_time = 1;

        if(quit){
          break;
        }
        pause();
      }
      original = 0;
    }

    // activate stealth mode
    if(mode == IPV6_STEALTH_MODE && !quit){
      int t, i = 0;
      struct link_info *tmp;
      struct link_list *tmp_list = NULL;

  // store original MACs
      if(first_time){
        list = get_ifinfo();
        if(list == NULL){
          HIP_ERROR("big loop: get_ifinfo failed\n");
          exit(EXIT_FAILURE);
        }
        first_time = 0;  //original values are store only first time
        default_ids = 0; //original values are changed
      }
      tmp_list = get_ifinfo();
      if(tmp_list == NULL){
        HIP_ERROR("big loop: get_ifinfo failed\n");
        exit(EXIT_FAILURE);
      }
      tmp = tmp_list->first;
      for(i; i < tmp_list->size; i++){
        bring_ifdown(tmp->name);
        set_new_mac(tmp->name, tmp->hw);
        set_ifup(tmp->name);
        tmp = tmp->next;
      }
      clean_link_list(tmp_list);
```

```
    if(quit){
      break;
    }

    /* sleep */
    t = sleep(cycle);
  }
  // activate normal mode
  else if (mode == IPV6_PUBLIC_MODE && !quit){

    if(quit){
      break;
    }
    //suspend the process until a signal arrives
    pause();
  }
  // exit loop
  else{
     break;
  }
}
```

Bibliography

[1] Andreasson, O. Ipsysctl tutorial 1.0.4. At `http://ipsysctl-tutorial.frozentux.net/ipsysctl-tutorial.html`, 2002. Referenced: 12th January 2006.

[2] anon. ip-sysctl.txt v 1.20. At Linux source tree linux/Documentation/networking/ip-sysctl.txt, December 2001.

[3] Arkko, J., Nikander, P., and Näslund, M. Enhancing Privacy with Shared Pseudo Random Sequences (preliminary version). to appear in *Security Protocols, 13rd International Workshop*, Cambridge, April 20-22 2005.

[4] David M. Goldschlag, M. G. R., and Syverson, P. F. Hiding Routing Information. In *Information Hiding* (Cambridge, U.K., May 30 - June 1 1996), R. J. Anderson, Ed., vol. 1174 of *Lecture Notes in Computer Science*, Springer, pp. 137–150.

[5] Debian Project. Net-tools 1.60. At `http://packages.debian.org/testing/net/net-tools`. Referenced: 10th November 2005.

[6] Draves, R. Default Address Selection for Internet Protocol version 6 (IPv6). RFC 3484, The Internet Engineering Task Force, February 2003.

[7] Gilligan, R., Thomson, S., Bound, J., McCann, J., and Stevens, W. Basic Socket Interface Extensions for IPv6. RFC 3493, The Internet Engineering Task Force, February 2003.

[8] Griffin, I., and Nelson, J. Linux Network Programming, Part 2. *Linux Journal 1998*, 47 (March 1998).

[9] Haddad, W., and Nordman, E. Privacy terminology, draft-haddad-alien-privacy-terminology-00. Internet-draft, The Internet Engineering Task Force, October 2005. Expires: March 2006.

[10] Haddad, W., Nordman, E., Dupont, F., Bagnulo, M., and Patil, B. Privacy for Mobile and Multi-homed Nodes: MoMiPriv Problem Statement, draft-haddad-momipriv-problem-statement-02. Internet-draft, The Internet Engineering Task Force, October 2005. Expires: April 2006.

[11] Hinden, R., and Deering, S. IP Version 6 Addressing Architecture. RFC 2373, Internet Engineering Task Force, July 1998.

[12] Huitema, C., and Carpenter, B. Deprecating Site Local Addresses. RFC 3879, Internet Engineering Task Force, September 2004.

[13] IEEE. Guidelines for 64-bit Global Identifier (EUI-64) Registration Authority. At `http://standards.ieee.org/regauth/oui/tutorials/EUI64.html`, March 1997. Referenced 8.12.2005.

[14] Johnson, M. K., and Troan., E. W. *Linux Application Development*, 2nd ed. Addison Wesley Professionals, 2004.

[15] Kernel.Org Organization, Inc. At `http://www.kernel.org/git/?p=linux/kernel/git/torvalds/linux-2.6.git;a=%summary`, 2006. Referenced: 4th January 2006.

[16] Komu, M. Application Programming Interfaces for the Host Identity Protocol. Master's thesis, Helsinki University of Technology, December 2004.

[17] Matos, A., Santos, J., Girao, J., Liebsch, M., and Aguiar, R. Host Identity Protocol Location Privacy Extension, draft-matos-hip-privacy-extension-01. Internet-draft, The Internet Engineering Task Force, March 2006. Expires: September 2006.

[18] Moskowitz, R., and Nikander, P. Host Identity Protocol Architecture, draft-ietf-hip-arch-03. Internet-draft, The Internet Engineering Task Force, August 2005. Expired: February 2006.

[19] Moskowitz, R., Nikander, P., Jokela, P., and Henderson, T. Host Identity Protocol, draft-ietf-hip-base-05. Internet-draft, The Internet Engineering Task Force, March 2006. Expires: September 2006.

[20] Narten, T., and Draves, R. Privacy Extension for Stateless Address Autoconfiguration in IPv6. RFC 3041, Internet Engineering Task Force, January 2001.

[21] NARTEN, T., DRAVES, R., AND KRISHNAN, S. Privacy Extensions for Stateless Address Autoconfiguration in IPv6, draft-ietf-ipv6-privacy-address-v2-04. Internet-draft, The Internet Engineering Task Force, May 2005. Expired: November 2005.

[22] NARTEN, T., NORDMARK, E., AND SIMPSON, W. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461, The Internet Engineering Task Force, December 1998.

[23] NIKANDER, P., ARKKO, J., AND HENDERSON, T. End-Host Mobility and Multihoming with the Host Identity Protocol, draft-ietf-hip-mm-03. Internet-draft, The Internet Engineering Task Force, February 2006. Expires: August 2006.

[24] NORDMARK, E., CHAKRABARTI, S., AND LAGANIER, J. IPv6 Socket API for Address Selection, draft-chakrabarti-ipv6-addrselect-api-03. Internet-draft, The Internet Engineering Task Force, July 2005. Expired: January 2006.

[25] PEKKA NIKANDER, JUKKA YLITALO, J. W. Integrating Security, Mobility, and Multi-homing in a HIP Way. In *Proceedings of Network and Distributed Systems Security Symposium (NDSS'03)* (San Diego, CA, February 2003), Internet Society, pp. 87–99.

[26] THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC. *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.* 3 Park Avenue, New York, NY 10016-5997, USA, 2001.

[27] THOMSON, S., AND NARTEN, T. IPv6 Stateless Autoconfiguration. RFC 2462, Internet Engineering Task Force, December 1998.

[28] USAGI. At `http://www.linux-ipv6.org/ml/usagi-users/msg03016.html`. Referenced: 19th August 2005.

[29] VIEGA, J., GIROUARD, Z., AND MESSIER, M. *Secure Programming Cookbook for C anc C++*, 1th ed. O'Reilly, 2003. Online `http://safari.oreilly.com/?XmlId=0-596-00394-3`.

[30] WATSON, D. Linux Daemon Writing HOWTO. At `http://www.linuxprofilm.com/articles/linux-daemon-howto.html`, May 2004. Referenced: 9th November 2005.

[31] WESTIN, A. F. Privacy and Freedom, 1970. Atheneum, New York.

[32] YLITALO, J., AND NIKANDER, P. A new Name Space for End-Points: Implementing secure Mobility and Multi-homing across the two versions of IP. In *Proceedings of The Fifth European Wireless Conference, Mobile and Wireless Systems beyond 3G* (Barcelona, Spain, February 24-27 2004), pp. 435–441.

[33] YLITALO, J., AND NIKANDER, P. BLIND: A Complete Identity Protection Framework for End-points. In *Proceedings of the Twelfth International Workshop on Security Protocols* (Cambridge, GB, April 26-28 2004).