Implementation of IPv6 in Internet Key Exhange version 2

Marko Salkić, Stjepan Groš, Vlado Glavinić
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing
Address: Unska 3, 10000 Zagreb, Croatia
Telephone: +385 1 6129-935 E-mail: marko.salkic@fer.hr

Summary – IKEv2 is a protocol for key exchange in Internet security architecture. We have implemented first, stripped-down version of IKEv2 that supported only IPv4 addresses. The implementation had some provisions that have been introduced in order to make later implementation of IPv6 easier. Currently, we are extending IKEv2 implementation with missing functionality in order to make it fully functional. In this paper we describe implementation details of IPv6 introduction into IKEv2 protocol implementation.

I. INTRODUCTION

IPv4 address architecture [1] is currently dominating on the Internet. This version has been in use for over 20 years and during this period of time many deficiencies were discovered and identified. This led to a new version, IPv6 architecture [2], that has been in active development for over a decade. There are many differences between two address architectures, the most important being larger address space, i.e. 32 bits vs. 128 bits.

The other important difference is mandatory implementation of security services on IP level in IPv6. Security services are described in a set of documents: a base one defining architecture[3], and additional set that specifies various protocols. These security services are known under the name IPsec. Implementation of IPsec is mandatory for all IPv6 compliant nodes, while for IPv4 it is optional. Still, many IPv4 compliant nodes also have IPsec implementation included. We are implementing *IKEv2 daemon*, key exchange protocol that is a part of the IPsec architecture. The first version supported only IPv4 but in the new version IPv6 is also supported.

This paper describes implementation process and differences between IPv4 and IPv6. It is structured as follows: firstly, in the second section we give a basic overview of IKEv2 protocol. Then, in the third section, we give an overview of IPv6 addresses and specifics of the IPv6 socket API in contrast to IPv4 socket API. Section four gives us an overview of handling IPv4 addresses in IKEv2 daemon, and then, an introduction of IPv6 addresses into daemon is described. Finally, the paper is concluded in section five, followed by a list of references in section six.

II. IKEv2 PROTOCOL

IP security (IPsec) is a suite of security protocols and open standards developed by the Internet Engineering Task Force (IETF) for secure transmission over unprotected networks, such as Internet. IPsec is required for IPv6 and optional for IPv4, but has not yet been widely deployed. Using cryptographic security services, IPsec provides confidentiality, data integrity, authenticity and availability. Setting the parameters of an IPsec security association (SA) manually does not scale well [4], so another protocol – Internet Key Exchange (IKE) – is used to automate and enhance negotiation of parameters and allow additional features. In its first version, IKE was a complex and somewhat flawed protocol that has been greatly simplified and improved in the new version, Internet Key Exchange version 2 (IKEv2).

IKE performs mutual authentication of peers and establishes IKE security association (SA) that includes shared secret information used to establish additional SAs for Encapsulating Security Protocol (ESP) and Authentication Header (AH). IKE negotiates keys, cryptographic and other parameters for securing data that SAs carry. SAs for ESP and/or AH that are set up through IKE SA (IKE_SA) are called “CHILD_SAs”. All IKE communication is based on pairs of request/response messages called exchanges. First two exchanges that establish an IKE_SA are called IKE_SA_INIT and IKE_AUTH. A minimum of four messages (two exchanges) is required to successfully establish IKE_SA and first CHILD_SA. Subsequent CREATE_CHILD_SA (for creating additional SAs) and INFORMATIONAL (for deleting SAs, error reports and other housekeeping) exchanges may follow in any number and order.

First request/response (IKE_SA_INIT) negotiates security parameters, nonces and Diffie-Hellman values for IKE_SA.

Second exchange (IKE_AUTH) authenticates first exchange by proving knowledge of shared secret in sent identities, and sets up a first AH and/or ESP CHILD_SA.

IPsec functions in two main modes: transport and tunnel mode. End-to-end communication is based on transport mode where only IP payload is encapsulated. In tunnel mode, the whole IP packet is encapsulated, thus both headers contain IP addresses.

Secured communication between Alice and Bob is defined by two sets of rules: what to protect and how to protect it. Those rules are contained in operating system databases: Security Policy Database (SPD), and Security Association Database (SAD or SADB). In SPD there are entries which consist of traffic selectors and actions (what to protect). Traffic selectors define packets based on their IP addresses, transport protocols, ports, etc. Set of security parameters of every connection specifying crypto algorithms and keys (how to protect) is called a Security Association (SA) and is a part of SADB. Each SA is associated by Security Parameters Index (SPI), and there is one SA for each direction of every connection. In other words, for every connection there is a pair of SA-s and SP-s (for every direction).

Should Alice request communication with Bob, security policy is chosen based on traffic selectors in SPD. Once consulted, the SPD defines three actions based upon a traffic match, as identified by the following selectors:

- DISCARD Do not let the packet in or out.
- BYPASS Do not apply or expect security.
- IPSEC Apply security protection

On Figure 1, the sequence of establishing a SA is shown. Alice has something to send to Bob (1). Before sending data, kernel identifies SPD that requires protection of the traffic. Since SADB on Alice's side is empty, and kernel needs protection parameters, IKE daemon is contacted with the request to provide necessary parameters (2). This initiates communication between IKEv2 daemons on Alice's and Bob's hosts that results in authentication, authorization and, finally, necessary keying material (3). Once negotiated, a new entry is made to SADB with appropriate SPI numbers (3, 4), IP addresses, actions, and other parameters thereby establishing a SA. Finally, the communication restarts and data is delivered to the Bob's application (5).

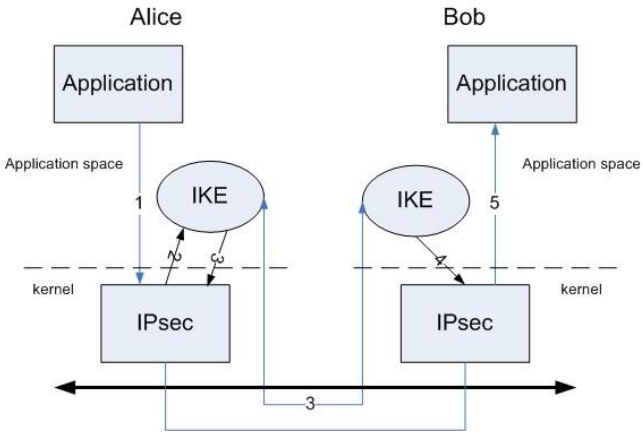

Figure 1. Communication between Bob'S and Alice's IKE

III. IPv6 AND SOCKET API

By 1996, years after the IPv4 address exhaustion problem had been identified, a series of RFCs (starting with RFC 2460) were released defining IPv6. Its global implementation has been somewhat slowed by the introduction of classless inter-domain routing (CIDR) [5] and Network Address Translation (NAT) [6]. However, its gradual implementation through isolated IPv6 “islands” is to be expected.

The main difference between IPv4 and IPv6, and the primary reason for the implementation of the new protocol is the address size: 32-bit for IPv4 versus 128-bit for IPv6 [7]. 128-bit address size guarantees, in effect, unlimited address space in any foreseeable future. The next important improvement is a simplified header: it is now of a fixed size, with optional extension headers. Other options can be found within extension headers, including AH and ESP. As a consequence of routing simplification, fragmentation is not allowed anywhere other then in the sending host. IPv6 addresses are scoped [7] so they could be link-local, site-local, organization, global, or other scopes at this time undefined. IPv6 allows stateless auto-configuration [8] by using several new features. Firstly, a temporary link-local address typically based on a MAC address is formed, until the characteristics of the network are discovered and a permanent address is obtained. Other features include: better multicasting, jumbograms, etc.

In order to enable applications to operate with IPv6 addresses, extensions to the existing Application Interface Protocol (API) have been made [9]. Along with keeping changes minimal where possible, some entirely new functions have been introduced. Generally speaking, IPv4 and IPv6 addresses are not compatible so it is essential to use API functions that hide differences as much as possible. `AF_INET6` is a designation of the new address family name that is used, for example, as a first argument to the `socket()` function.

In the socket interface, a different protocol-specific structure sockaddr_in6 is used:

```
struct sockaddr_in6 {
     sa_family_t    sin6_family;
     in_port_t sin6_port;
     uint32_t  sin6_flowinfo;
     struct in6_addr
     sin6_addr;
     uint32_t  sin6_scope_id;
}
```

The sin6_family field corresponds to the sa_family field in a protocol-independent structure sockaddr. The value of sin6_family has to be AF_INET6 to identify the structure as sockaddr_in6. Any protocol-specific structure is cast into generic structure sockaddr before it is used with socket functions. Similar to sockaddr_in (the IPv4 version of sockaddr_in6), sin6_port contains the 16-bit TCP or UDP port number in network byte order. Fields sin6_flowinfo and sin6_scope_id are IPv6 specific, as specified by the new architecture. The sin6_addr field is of type in6_addr and is large enough to hold a 128-bit address. Special consideration is needed when specifying the size of a socket data structure. The sizes of structures sockaddr and sockaddr_in are equal, while sockaddr_in6 is larger, so it is wrong to use the generic structure sockaddr as a measure of the structure's size. Using functions bind(), connect(), sendmsg() and sento() with IPv6 addresses is analogous with IPv4 addresses: the difference is in the address family (AF_INET6) and in the structure used for passing over the address (sockaddr_in6). The same is true for functions accept(), recvfrom(), recvmsg(), getpeername() and getsockname(). Furthermore, the new API requires the use of functions such as: inet_pton() and inet_ntop() (instead of inet_addr() and inet_ntoa()) for address conversion, getaddrinfo(), getipnodebyname(), getipnodebyaddr() (instead of gethostbyname() and gethostbyaddr()) for nodename-to-address translation and vice versa, etc. The aforementioned functions provide backward compatibility while introducing IPv6 support at the same time. Ancillary data that is used in IKEv2 to exchange optional information like packets' source and destination addresses has been adjusted to the new API [4, 6, 5, 3]. To receive any optional information, application must set socket to do so with setsockopt(). The parameter needed is IPV6_RECVPKTINFO of type IPPROTO_IPV6.

Having in mind the forthcoming IPv6 protocol, new applications should work in a mixed IPv4/IPv6 environment. Essentially, the performance and reliability of applications should not depend on such variables. Linux kernel's dual stack allows IPv4 and IPv6 stacks to coexist [10], enabling applications to use both protocol versions simultaneously. The API also provides ability for IPv6 applications to interoperate with IPv4 applications, and vice versa. This feature uses IPv4-mapped IPv6 addresses that embeds IPv4 into IPv6 addresses. IPv4-mapped IPv6 addresses are written as follows:

```
::FFFF:<IPv4-address>
```

If an application is listening on an IPv6 socket, and if a IPv4 packet is to arrive (provided socket is not set to accept IPv6 packets only), then the kernel converts IPv4 into IPv4-mapped-IPv6 address. The actual communication between the nodes takes place using IPv4 datagrams, but the application doesn't need to be aware of that. If, however, the application requires the knowledge of how the actual communication takes place, then macro function IN6_IS_ADDR_V4MAPPED can be utilized.

IV. IPv6 IMPLEMENTATION IN IKEv2

IKEv2 implementation was designed to allow simultaneous use of both IPv4 and IPv6 addresses from the beginning. This is achieved via generic structure netaddr, and a set of functions that hide differences between addresses from the majority of the code. The netaddr structure has the following members:

```
struct netaddr {
     union {
          struct sockaddr sa;
          struct sockaddr_in sin;
          struct sockaddr_in6 sin6;
          struct sockaddr_ll sll;
     };
     guint8 prefix;
     gint refcnt;
}
```

The other purpose of introduction of special structure is the necessity to also include network masks with addresses (prefix), which standard socket structures do not allow. Structure netaddr uses union of different types of socket structures, thus reserving sufficient amount of memory for any one of them, i.e. the most memory-consuming one. Accordingly, the type of socket structure is transparent to the programmer. This prevents code duplication because it would be necessary to separately treat all socket structures were it not for the unified structure.

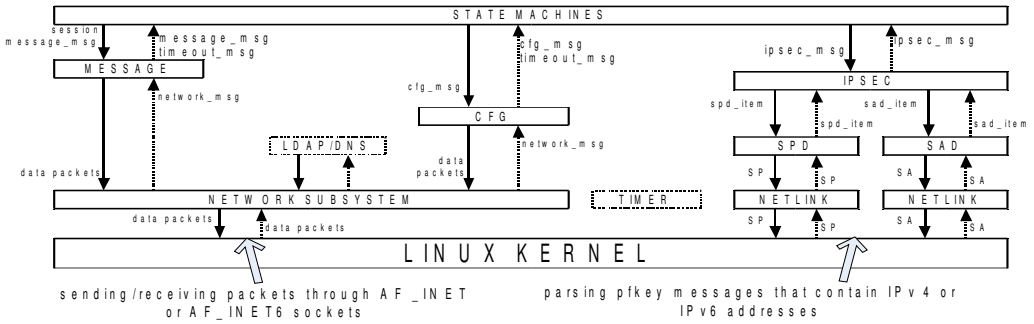

Figure 2. Architecture of IKEv2 implementation with IP version dependent points

Although the majority of the code is independent from the actual version of IP address, there are situations where it is impossible to treat them the same. On Figure 2 architecture of IKEv2 daemon is shown, and interfaces that depend on particular version of IP address are identified. Acquiring SADB messages from the kernel that contain traffic selectors is done via `pfkey` socket, as shown on Figure 2. When receiving or sending `pfkey` messages, IP addresses have to be parsed from `pfkey` messages into `netaddr` structures or transformed from `netaddr` structures when sending `pfkey` messages to the kernel.

In the first phase of the IKEv2 project we decided to support only IPv4 addresses due to the time constraints. IPv6 addresses have been added during the second phase of IKEv2 project, which is currently in progress. Due to the prepared architecture, the addition has passed relatively easy. However, we've encountered problems.

Using transitional mechanisms “Bump-In-the-Stack” (BIS) [11] and “Bump in the API” (BIA) [12] is not applicable to IKEv2 because some parts depend profoundly on IP addresses (e.g. traffic selectors). Besides that, it is highly inadvisable to apply BIS or BIA as a patch to IPv4 applications when it is possible to rewrite them.

When we've started implementing IPv6 support into IKE, the following question has been self-inflicted: with all considering, should the application be written to deal with IPv6 addresses only? Structure `netaddr` is indifferent of the IP address version, a set of functions directly dealing with IP addresses are concealing addresses' versions, so the rest of the code would not have been afflicted by the changes. Focusing on one version would simplify and minimize the code as well as make it easier to understand and maintain. It is possible to ignore IPv4 addresses as such by using kernel’s ability to transform all IPv4 addresses into IPv4-mapped IPv6 addresses, thus conserving interoperability with IPv4 nodes. However, IP addresses are an integral part of traffic selectors.

As mentioned earlier, traffic selectors are used to define traffic to be protected, discarded or allowed as defined in SPD. The structure of traffic selectors is defined by an address range (IPv4 or IPv6), a port range and protocol IDs. IKEv2 allows traffic negotiating where the responder is allowed to choose a subset of the traffic proposed by the initiator. Calculating intersections is based on functions that compare IP addresses and their families. One possible scenario is that kernel delivers an IPv4-mapped IPv6 address to be compared with an IPv4 address. Fortunately, the format of IPv4-mapped IPv6 address provides an easy way of doing that - just by comparing last 32 bits with regular IPv4 address. Similarly, the use of macro function `IN6_IS_ADDR_V4MAPPED` is required wherever it is important to identify an IPv4-mapped IPv6 address. This would, in effect, violate the principles of code encapsulation. Moreover, some consider that the use of IPv4-mapped IPv6 addresses is a security vulnerability [13]. On the other hand, we have already had a stable and a tested functionality and that was unwisely to discard, so it has been finally decided to keep the existing code and add IPv6 support in parallel. Consequently, a separate `AF_INET6` socket was added and restricted to accept IPv6 packets only. To prevent kernel from acquiring IPv4 packets on a IPv6 socket, socket option `IPV6_V6ONLY` is set using the `setsockopt()` function [9].

V. CONCLUSION

Although there is some dispute as to when a general acceptance of IPv6 is to be expected among applications, all new network applications should work with IPv6. Since IPsec is required for IPv6, and IKEv2 stands as an important part of it, including IPv6 in IKEv2 daemon is especially important and can, in a way, support its application. IPv6 is not expected to start dominating the Internet over night, so in transitional period, applications ought to work with IPv4 and IPv6. As a result, applications will be ready for pure IPv6 networks (or islands) after the transitional period is over. Kernel and the

new API provide a way for applications to support IPv4 and IPv6 in parallel, e.g. with socket dual binds. We have decided to keep the existing IPv4 code and adjust it for IPv6 support. Since the code is structured in modules and prepared for occurrences of IPv4 and IPv6 addresses in advance, it was not hard to adjust the network section and existing socket functionalities. Up until the second phase, IKE had been using functions compatible with the new API, with the exception of socket parameters. However, some dependencies have not been possible to mask. The result is a little bit more complex than it would have been had we decided to implement IPv6 support only, but the resulting code is more flexible for usage during the transitional period. Ideally, applications are not supposed to depend on IP version used, but as IKE is concerned, that is not entirely the case. Using kernel's mechanisms to translate any IPv4 in IPv4-mapped IPv6 address leads to some problems that can be avoided by separating sockets.

To conclude, modular architecture of network applications can save a lot of time and effort when transitioning to some other protocol, as it was the case with IPv6. Had not structures been prepared to hold IPv4 and IPv6 addresses from the beginning, the transition and rewriting the code would have been incomparably more difficult. Hopefully, this document has given some ideas how to make those transitions easier with applications similar to IKEv2.

VI. ACKNOWLEDGMENT

This work has been carried out within projects 036-0361994-1995 Universal Middleware Platform for Intelligent e-Learning Systems funded by the Ministry of Science and Technology of the Republic of Croatia, and IKEv2 Step2 project funded by Siemens Networks.

VII. LITERATURE

[1] Postel, J., “Internet Protocol”, RFC 791, September 1981

[2] Deering, S., Hinden R., “Internet Protocol, Version 6 (IPv6) Specification”, RFC 2460, December 1998

[3] Kent, S., Atkinson, R., “Security Architecture for the Internet Protocol”, RFC 2401, November 1998

[4] Kaufman, C., “Internet Key Exchange (IKEv2) Protocol”, RFC 4306, December 2005

[5] Fuller, V., Li, T., Yu, J., “Classless Inter-Domain Routing (CIDR)”, RFC 1519, September 1993

[6] Srisuresh, P., Egevang, K., “Traditional IP Network Address Translator (Traditional NAT)”, RFC 3022, January 2001

[7] Hinden, R., Deering, S., “IP Version 6 Addressing Architecture”, RFC 4291, February 2006

[8] Thomson, S., Narten, T., “IPv6 Stateless Address Autoconfiguration”, RFC 2462, December 1998

[9] Gilligan, R. et al., “Basic Socket Interface Extensions for IPv6”, RFC 3493, February 2003

[10] Nordmark, E., Gilligan, R., “Basic Transition Mechanisms for IPv6 Hosts and Routers”, RFC 4213, October 2005

[11] Tsuchiya, K. et al., “Dual Stack Hosts using the 'Bump-In-the-Stack' Technique (BIS)”, RFC 2767, February 2000

[12] Lee, S. et al., “Dual Stack Hosts Using 'Bump-in-the-API' (BIA)”, RFC 3338, October 2002

[13] Davies, E. et al., “IPv6 Transition/Co-existence Security Considerations”, draft-ietf-v6ops-security-overview-06 (work in progress), October 2006