IP 계층에 통합된 IPsec 엔진 구현 방법

박소희, 정지훈, 나재훈*
*한국전자통신연구원, 정보보호기술연구본부

The Implementation of IPsec Engine integrated IP Layer

So-hee Park, Ji-hoon Jeong, Jae-hoon Nah*
*Information Security Technology Division, .ETRI.
E-mail : {parksh, jihoon, jhnah}@etri.re.kr

요 약

인터넷의 활용이 급속하게 증가하여 인터넷에서의 정보보호에 대한 필요성이 대두되면서 표준화된 인터넷 정보보호 프로토콜인 IPsec이 등장하게 되었다. 이러한 IPsec은 현재 여러 가지 플랫폼에서 구현되고 있으며 이러한 구현은 일반적으로 IP 계층에 통합하는 방법, BITS, BITW 중 하나의 방법론을 선택하고 있다. 본 논문에서는 IPsec 구현 방법론을 간단히 살펴보고 이들의 장단점을 분석하여 이 중 가장 효율적이라 생각되는 IP 계층에 IPsec을 통합하는 방법을 선택하여 구현하였다. 이에 본 논문은 공개된 운영체제인 리눅스 커널 상에서 IPsec을 구현하기 위해 리눅스 커널의 IP 계층 및 소켓 버퍼 구조를 분석하고 정보보호 정책(SPDB)과 SADB와 연동되는 IPsec 엔진을 IP 계층에 통합하여 구현하는 방법을 제안한다.

1. 서론

인터넷의 이용이 보편화되면서 보안성 결핍의 문제를 해결하기 위한 노력은 계속되고 있다. 이에 IETF IPsec WG에서는 1993년 6월부터 인터넷 정보보호에 대한 기본 구조를 연구하여 현재 IPsec 아키텍처를 정의한 RFC2401을 비롯한 18개의 RFC를 기술하였다[1]. IPsec은 IPv4에서는 선택 기능으로 IPv6에서는 필수 기능으로 채택된 프로토콜로 기존 특정 응용 프로그램에 의존한 정보보호 서비스를 IP 계층에서 패킷 단위로 보호하기 위해 제안되었으며, 인증을 위한 헤더인 AH(Authentication Header)와 기밀성을 위한 헤더인 ESP(Encapsulation Security Payload) 두 확장헤더 및 키 교환을 위한 IKE(Internet Key Exchange)를 이용한다[2,3,4].

IP 계층에서 정보보호 서비스를 제공하는 IPsec은 일반 사용자에게 투명한 상태로 처리되며, 응용 계층 및 트랜스포트 계층의 모든 프로토콜에 공통된 정보보호 서비스를 제공할 수 있어 한 호스트 내에서는 일관된 방식의 정보보호 서비스 설정이 가능하다. 또한 특정 알고리즘에 국한되지 않기 때문에 새로운 암호화 기술 적용이 용이하다.

IPsec이 제공할 수 있는 정보보호 서비스는 접근 제어, 데이터 원적지 인증, 비연결형 인증, 데이터 기밀성, 재현 공격 방지, 제한된 트래픽 흐름 기밀성이며 ESP는 이러한 서비스를 모두 제공할 수 있으나, AH는 데이터 기밀성 및 제한된 트래픽 흐름 기밀성은 제공하지 않아 필요한 보안 수준에 적절하게 두 확장헤더를 선택하여 사용하면 된다.

현재 IPsec은 리눅스, FreeBSD, WINDOWS 2000 등 여러 가지 플랫폼에서 구현되고 있으며, 리눅스 기반의 FreeS/Wan, FreeBSD 기반의 KAME 등 공개된 프로젝트도 존재한다[5,6]. 이러한 구현은 일반적으로 IP 계층에 통합하는 방법과 BITS, BITW 중 하나의 방법론을 채택하고 있다. 첫 번째 구현 방법은 IPsec이 IP계층에서 제공되는 서비스라는 측면에서 가장 원칙적인 방법으로 IP 헤더 생성과 함께 AH/ESP 처리를 하는 방법이며, BITS(Bump-in-the-Stack)는 이와는 달리 기존 IP 계층과 Data Link 계층 사이에 IPsec을 끼워넣는 방법이고, 마지막 BITW(Bump-in-the-Wire)는 물리적인 인터페이스 카드 내에 IPsec 엔진을 구현하는 방법이다. 이에 본 논문에서는 이들 구현 방법의 장단점을 비교

분석하고 이 중 가장 효율적이라 생각되는 IP 계층에 통합되는 방법으로 IPsec 엔진을 리눅스 커널 상에서 구현하는 방법을 제안하고자 한다. 리눅스 커널 상에서 IPsec 엔진을 구현하므로 리눅스 커널의 소켓 버퍼 구조 및 IP 계층의 네트워크 흐름을 분석하고 IPsec 엔진이 구동하기 위해 필요한 정보보호 정책 및 SADB와의 연동에 관해서도 살펴본다.

2. 일반적인 IPsec 엔진의 구현 방법론

RFC 2401에는 IPsec의 구현 방법을 IP 계층에 통합하는 방법, BITS, BITW 세 가지로 분류해 놓고 있다. 이러한 세 가지 방법론은 각각의 장단점을 가지고 있으며 이 장에서는 이러한 방법론에 대하여 논하기로 한다.

2.1 IP 계층에 통합하는 방법

IPsec은 원칙적으로 IP 계층에서 제공되는 보안 프로토콜이므로 그림 1과 같이 IP에 계층에 통합되는 것이 가장 바람직한 방법이라고 볼 수 있다. IP 계층에서 정책과 보안 협상을 조회하고 IPsec 적용 여부를 판단할 수 있으며, 기존 IP 헤더 생성과 함께 AH/ESP 처리를 할 수 있다. 이 구현 방법은 완전하게 IP 계층에 통합된다. 보안호스트와 보안게이트웨이에 모두 적용이 가능하다.

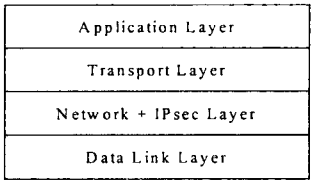

그림 1. IP 계층에 통합하는 방법

2.2 BITS(Bump-in-the-Stack)

그림 2와 같이 IPsec 엔진을 IP 계층과 Data Link 계층 사이에 끼워 넣는 방법인 BITS 방법은 보안 호스트에만 가능한 것으로 다른 계층의 수정없이 IPsec을 처리할 수 있어 구현이 용이하다.

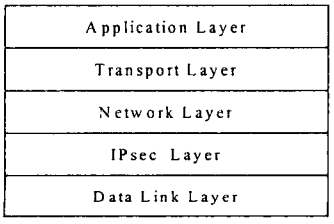

그림 2. BITS(Bump-in-the-Stack)

2.3 BITW(Bump-in-the-Wire)

Outboard crypto processor를 사용하여 물리적인 인터페이스 카드가 있는 디바이스 내에 IPsec을 구현하는 방법으로 그림 3과 같이 인터페이스를 통과하는 모든 패킷에 IPsec이 적용된다. 일반적으로 IPsec이 구현된 디바이스는 라우팅과 같은 다른 역할을 수행하는 것이 아니라 단순하게 패킷에 대한 보안 서비스만을 제공한다. 보안호스트와 보안게이트웨이에 모두 적용 가능하다.

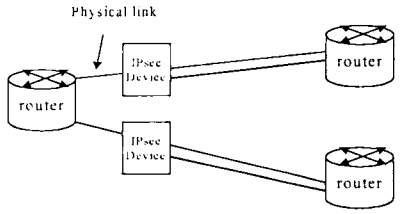

그림 3. BITW(Bump-in-the-Wire)

2.4 비교 및 분석

위에서 설명한 구현 방법론은 크게 확장성, 구현의 용이성 및 성능상의 측면에서 비교 분석할 수 있다.

IPsec 엔진을 IP 계층에 통합하는 방법은 IP 소스 코드 내에 삽입되므로 IP 소스 코드가 변경되면 IPsec 코드도 수정되어야 하기 때문에 확장성이 떨어지며 리눅스와 같이 outbound IP 패킷을 처리하는 함수가 나누어져 있는 경우 각각의 함수에 모두 IPsec을 처리하는 코드를 삽입해야 하는 구현상의 번거로움은 존재하나, 한번의 IP 패킷 처리로 IPsec 처리까지 모두 이루어지므로 이중적인 fragmentation과 같은 IP 패킷 처리는 존재하지 않는다. 이에 비해 BITS 방법은 기존의 리눅스 커널 소스 코드와 의존성이 없어 IPsec 코드에 대한 확장성이 보장되며, 하나의 함수에서 호출되므로 IPsec 처리를 위한 코드 구현이 용이하다. 그러나, IPsec 패킷에 대한 이중적인 IP 패킷 처리가 존재하므로 성능상의 문제점이 발생할 수 있다. BITW 방법은 기존 시스템의 인터페이스에 디바이스를 장착하는 것만으로 IPsec을 지원할 수 있어 기존 시스템 이용의 효율성은 높으나 이는 IPsec을 지원하는 시스템으로 넘어가기 위한 과도기적인 것으로 장기적인 관점에서 바람직하지 않다.

3. 리눅스 커널 분석

공개된 운영체제인 리눅스를 이용하여 IPsec 엔진을 구현하므로 이 장에서는 리눅스 커널 2.2.12의 네트워크 영역을 분석한다. 리눅스에서 네트워크 데이터를 다루기 위해 사용되는 소켓 버퍼 sk_buff와

일반적인 리눅스 커널의 네트워크 흐름을 살펴보는 데 설명의 편의상 TCP 연결상의 루틴만을 기술한다.

3.1 소켓 버퍼(Socket Buffer)

리눅스는 많은 네트워크 계층을 가지고 있어 각각의 서비스에서 이를 사용한다. 프로토콜 계층은 전송하는 데이터의 프로토콜 헤더와 트레일러를 각 계층마다 특정 헤더와 트레일러를 찾아 추가하고 제거해야 하는 문제점이 있다. 프로토콜 계층마다 버퍼를 복사하여 사용할 수도 있으나 이는 매우 비효율적이므로 이를 해결하기 위해 sk_buff라는 소켓버퍼를 사용한다. 소켓 버퍼는 포인터를 가지고 있어 각 프로토콜 계층에서 표준 함수를 통해 응용 프로그램 데이터를 다룰 수 있도록 한다. 그림 4는 소켓 버퍼의 자료 구조를 보여준다. 각 소켓 버퍼는 자신과 연관된 데이터 블록을 가지고 있고 이들을 다루는데 필요한 4개의 포인터 head, data, tail, end를 가지고 있다. head와 end는 메모리가 할당될 때 결정되는 것으로 데이터 영역의 시작점과 끝점을 가리키고, data와 tail은 현재 프로토콜 데이터 시작점과 끝점을 가리키는 것으로 프로토콜 계층에 따라 달라지는 값이다.

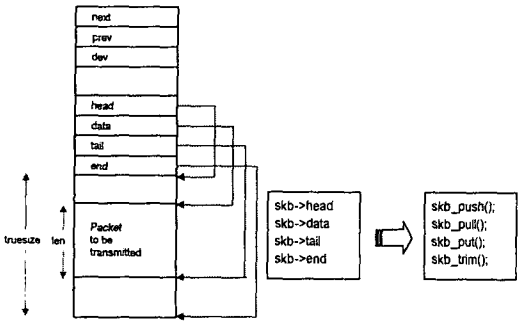

그림 4. sk_buff의 자료 구조

길이를 나타내는 항목으로 len과 truesize 두 개가 있으며, 이들은 각각 현재 프로토콜 패킷의 길이와 상대적인 데이터 버퍼의 전체 길이를 나타낸다. sk_buff를 다루는 코드는 응용 프로그램 데이터에 프로토콜 헤더와 트레일러를 붙이고 제거하는 표준적인 방법을 다음과 같이 제공한다.

(1) skb_push() : 프로토콜 헤더나 데이터를 데이터 시작부분에 추가하는데 사용하는 것으로 data 포인터를 시작쪽으로 이동하고 len을 증가시킨다.

(2) skb_put() : 프로토콜 트레일러나 데이터를 데이터 끝부분에 추가하는데 사용하는 것으로 tail 포인터를 끝쪽으로 이동하고 len을 증가시킨다.

(3) skb_pull() : 수신한 패킷의 데이터 시작부분이나 프로토콜 헤더를 제거하는데 사용하는 것으로 data 포인터를 끝쪽으로 이동하고 len을 감소시킨다.

(4) skb_trim() : 수신한 패킷의 데이터 끝부분이나 프로토콜 트레일러를 제거하는데 사용하는 것으로 tail 포인터를 시작쪽으로 이동하고 len을 감소시킨다.

3.2 일반적인 리눅스 커널의 네트워크 흐름

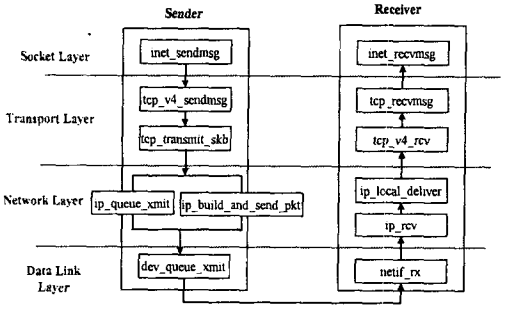

그림 5. 리눅스의 Normal TCP/IP 흐름도

그림 5는 일반적인 리눅스 커널의 TCP/IP 네트워크 흐름도를 계층별로 도식화한 것이다. 그림 5와 같이 inet_sendmsg()를 통하여 INET 계층에서 TCP 계층으로 패킷이 내려오며, TCP 계층에서는 tcp_v4_sendmsg()와 tcp_transmit_skb()를 통하여 IP 계층으로 패킷을 내려보낸다. 이렇게 IP 계층으로 내려온 IP 패킷은 ip_queue_xmit()과 ip_build_and_send_pkt() 함수를 통해 IP 패킷 처리가 되고 dev_queue_xmit()을 통해 디바이스 계층으로 내려간다. IP 계층에서 송신측이 패킷을 보낼 때는 ip_queue_xmit()이 호출되고, 수신측이 받은 패킷에 대한 응답을 보낼 때는 ip_build_and_send_pkt()이 호출된다. 디바이스 계층으로 내려가서 전송된 패킷은 수신측 디바이스 버퍼에 쌓여있다가 netif_rx()를 통해 IP 계층으로 올라가고 ip_rcv()와 ip_local_deliver()가 순차적으로 처리되어 IP 패킷 처리가 완료된 후 tcp_v4_rcv()와 tcp_recvmsg()

에 의해 TCP 처리가 되고 inet_recvmsg()에 의해 응용 계층으로 올라간다.

4. IP 계층에 통합된 IPsec 엔진 구현 방법 제안

이 장에서는 2장에서 살펴본 구현 방법론 중 성능 측면에서 가장 효율적이라 생각되는 IP 계층에 통합하는 방법으로 IPsec 엔진을 구현하는 방법을 제안한다. IP 계층에서 IPsec 엔진을 통합하는 방법은 그림 6과 같은 흐름도로 나타낼 수 있다.

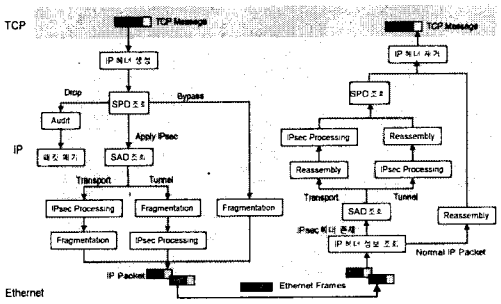

그림 6. IP 계층에 통합된 IPsec 처리 흐름도

4.1 Outbound Processing

IP 계층에서 IP 헤더 생성 및 옵션 처리가 모두 이루어지고 나면 이 단계에서 정책을 조회하고 정책이 "Bypass"이면 일반적인 IP 패킷 처리를 계속 수행하고, "Discard"이면 audit()을 수행하고 패킷을 폐기하며, "Apply IPsec"이면 IPsec 처리를 수행한다. "Apply IPsec"이 적용되면 적절한 SA를 조회하고 SA에 맞는 IPsec 처리를 수행한다. 트랜스포트 모드인 경우에는 패킷의 fragmentation이 일어나기 전에 IPsec 처리가 수행되어야 하고, 터널 모드인 경우에는 IP 패킷을 캡슐화하여야 하므로 fragmentation이 일어나고 난 다음에 IPsec 처리가 수행되어야 한다.

리눅스 커널의 IP 계층에서는 outbound 처리를 담당하는 ip_output.c와 보안 게이트웨이의 outbound 처리를 담당하는 ip_forward.c이 IPsec outbound processing과 관련된다. ip_output.c에서는 크게 4개의 함수가 존재하는데, 위에서 살펴본 ip_queue_xmit() 함수와 ip_build_and_send_pkt() 함수는 TCP가 상위 프로토콜일 때 사용되는 함수이며, ip_build_xmit() 함수와 ip_build_xmit_slow() 함수는 상위 프로토콜이 UDP이거나 ICMP일 때 호출된다. IP 계층에 통합된 IPsec 엔진을 구현하기 위해서는 이 네 가지 함수에 모두 IPsec 처리를 하는 루틴을 삽입하여야 한다. ip_forward()에서는 outbound 함수에서 패킷을 처리하는 동일한 방식으로 IPsec을 적용한다.

4.2 Inbound Processing

수신측에서는 netif_rx()를 통해 IP 계층으로 올라온 패킷은 reassembly 하기 전에 수신된 IP 패킷의 헤더 정보를 보고 IPsec이 적용된 패킷이면 SA를 조회해 보고 터널 모드이면 reassembly 하기 전에 트랜스포트 모드이면 reassembly 하고 난 후에 IPsec 처리를 한다. 마지막으로 정책을 조회하여 적용된 보안 협상이 적절한 것인지를 판단한다.

보안호스트는 그림 6과 같은 흐름도를 따라 수행하지만, 보안게이트웨이는 모두 조립된 IP 패킷을 받은 후 다음 호스트나 게이트웨이로 패킷을 포워드하는 기능만을 수행하므로 IPsec 처리가 된 패킷을 받은 후 도착지 주소가 자신인 것만을 처리하고 나머지는 포워드한다. 포워드할 때 정책을 조회하여 그림 6과 동일한 방법으로 IPsec을 적용하여 SA Bundle도 지원한다.

리눅스 IP 계층에서 inbound 처리를 담당하는 in_input.c에는 IP 패킷 자체의 에러를 검증하는 ip_rcv() 함수와 에러 검증이 끝난 패킷을 reassembly하고 상위 트랜스포트 계층으로 올려보내는 ip_local_deliver() 함수가 있는데, 보안호스트에서는 reassembly가 끝난 후에 IPsec 처리를 하므로 ip_local_deliver()에서 IPsec 처리가 일어나고 보안게이트웨이에서는 reassembly가 되기 전에 IPsec 처리가 되어야 하므로 ip_rcv()에서 IPsec 처리가 수행된다. 보안게이트웨이는 ip_rcv()에서 IPsec이 적용되어 있는 모든 패킷에 대해 IPsec 처리를 하는 것이 아니라 패킷의 도착지 주소가 자신인 것만 처리하고, ip_forward.c 내에 있는 ip_forward() 함수로 패킷을 보낸다.

5. 결론

본 논문에서는 리눅스 커널의 네트워크 측면을 분석하고 IPsec 엔진을 구현할 수 있는 일반적인 구현 방법론 각각의 장단점을 확장성과 구현 용이성 및 성능 측면에서 비교 분석하였다. 이 중 표준이 되는 RFC 2401에 가장 부합되는 방법론으로 IPsec 엔

진을 기존의 IP 계층에 통합하는 방법을 채택하여 실제 리눅스 커널 상에 구현하였다. 이에 구현에 필요한 흐름도를 제안하였고 이를 outbound와 inbound 측면에서 나누어서 살펴보았다. 이는 다른 방법에 비해 확장성과 구현 용이성은 떨어지나 성능 측면에서 우수한 특성을 보인다.

[참고문헌]

[1] S.Kent, R.Atkinson, "Security Architecture for the Internet Protocol", RFC2401, Nov. 1998.
[2] S.Kent, R.Atkinson, "IP Authentication Header", RFC2402, Nov. 1998.
[3] S.Kent, R.Atkinson, "IP Encapsulation Security Payload", RFC2406, Nov. 1998.
[4] H.Harney, C.Muckenhirn, "The Internet Key Exchange", RFC2409, Nov. 1998.
[5] FreeS/Wan form http://www.freeswan.org
[6] KAME form http://www.kame.net