碩士學位論文

Hacking-Free-Packet 필터를 이용한 침입탐지 시스템 성능 향상

濟州大學校 大學院

電氣電子工學科

金　性　潤

2002 年 12 月

Hacking-Free-Packet 필터를 이용한 침입탐지 시스템 성능 향상

指導教授 金　敬 植

金 性 潤

이 論文을 工學 碩士學位 論文으로 提出함

2002年 12月

金性潤의 工學 碩士學位 論文을 認准함

審査委員長 ______________印

委　　　員 ______________印

委　　　員 ______________印

濟州大學校 大學院

2002年 12月

Enhancement of IDS performance by using a Hacking-Free-Packet filter

Soung-Yun Kim
(Supervised by professor Kyung-Sik Kim)

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Engineering

2002. 12.

Department of Electrical and Electronic Engineering
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

목차

SUMMARY

The internet security industry has grown rapidly along with the increase in the number of security attacks during the past several yeas. Among the various security protection tools, Firewall and IDS(intrusion detection system) are most well-known. Those security tools need to handle a huge amount of network traffic from tremendously growing internet connectivity.

IDS must analyze all incoming packets to detect the signatures of intrusion. With a single packet to fail analyze, the IDS may lose a critical clue of intrusion and fail to detect an intrusion.

In this thesis, a system has been designed and implemented to overcome efficiently the bottleneck occurring in IDS. The system is developed at the layer of intermediate driver in MS Windows. Most of packet capture modules used in IDS is developed in the protocol driver layer. The proposed system is independent of packet capture modules of various IDS.

Since the dominant portion of internet traffic is web traffic, IDS need not to monitor major portion of the traffic by dropping the web traffic in the intermediate driver layer. To analyze receiving packets efficiently, a filter module is designed in the form CFG(control flow graph) model using ASL(audit specification language) based on BPF(berkley packet filter).

The proposed system can analyze all of packets existing in a fast ethernet network and filter Hacking-Free-Packets completely. The performance of IDS using the proposed system is improved by increasing the amount of web traffic due to heavy network traffic.

I. 서론

과거 몇 년간 인터넷 사용자의 폭발적인 증가와 함께 해킹에 의한 피해 사례 또한 급증하고 있다(이, 2001, 2001b). 이에 보안 관련 연구 및 제품 또한 급속도로 진전, 출시되고 있는 상황이며, 그 중에서도 방화벽이나 침입탐지 시스템은 이와 관련된 잘 알려진 분야라고 할 수 있다(김, 2001). 이러한 제품들은 인터넷 접속량이 늘어남에 따라 높은 네트워크 트래픽을 다룰 수 있어야 하는데, 이는 이들의 특성상 네트워크로 들어오는 모든 패킷에 대한 분석이 필요하기 때문이다. 네트워크의 모든 패킷을 분석하고, 그에 대응하기 위해서는 높은 오버헤드가 발생하게 되고, 결국 패킷 손실이나, 실시간 적인 대응을 할 수 없게 만든다(Willam, 1997). 실례로, Mirecom Lab 의 보고서에 의하면 기가비트 이더넷에서 986.94 Mbps 백그라운드 트래픽을 갖는 네트워크에서 침입탐지 비율은 44% 정도라고 한다(Mire, 2001).

이러한 한계점을 극복하기 위해, 하드웨어적으로 침입탐지 시스템과 연동하여 패킷 필터링을 할 수 있는 임베디드 시스템이 제안되어지고 있다(Jongwook, 2001). 또한 소프트웨어적인 측면에서는, 침입탐지 시스템의 탐지 알고리즘 분야(Snort)와 침입탐지 시스템의 패킷 캡쳐 모듈 자체의 처리속도를 높이기 위한 분야로, Pcap이라는 오픈소스 패킷 캡쳐 모듈이 개발, 진행되고 있다(Pcap, 2000). 패킷 캡쳐 모듈의 패킷 필터링을 위해서는 안정적이고, 빠른 BPF(berkley packet filter)가 사용되고 있으며(McCanne, 1993), 그 밖에도 BPF의 성능 향상을 위한 연구도 이루어 지고 있다(Guang, 1998). 소프트웨어적인 측면에서, 침입탐지 시스템의 탐지 알고리즘을 개선하거나 패킷 캡쳐 모듈을 교체하는 것은 기존의 시스템을 유지하기 어렵게 만들어, 다양한 침입탐지 시스템에 독립적이지 못하다.

이에 본 논문에서는 기존의 시스템을 유지하면서 높은 네트워크 트래픽에서의 침입탐지 시스템의 탐지 성능향상을 시킬 수 있는 시스템을 제안하고 있다. 이는 MS (Microsoft) Windows의 네트워크 드라이버 구조에 기인하는데, 대부분의 침입탐지 시스템의 패킷 캡쳐 모듈이 프로토콜 드라이버에서 구현되어 있음을 고려한 것이다 (MSDN, 2000). 제안된 필터 모듈은 인터미디엇 드라이버(intermediate driver)로 구

현되었으며, 이는 기본 네트워크 구조를 해치지 않고 패킷의 송수신에 관여할 수 있는 구조로, 이를 이용해 해킹과 무관한 패킷(Hacking-Free-Packet) 을 필터링 하여, 침입탐지 시스템의 탐지 성능을 향상시키고자 하였다. 이러한 구현의 장점은, 하드웨어적인 접근에 비해 비용이 적고, 다양한 침입탐지 시스템이 사용하고 있는 패킷 캡쳐 모듈에 독립적이라는 이점을 가지고 있다. 이를 위해 우선 인터미디엇 드라이버를 개발하였고, 개발된 인터미디엇 드라이버에 Hacking-Free-Packet을 필터링 할 수 있는 필터 모듈을 추가하였다. 필터 모듈은 ASL(Auditing Specification Language) (Guang, 1998)을 이용하여 CFG(control flow graph) 모델로 구현함으로서 처리 속도 및 안정성을 높였다.

본 논문의 구성을 살펴보면, II 장에서는 네트워크 보안을 위한 침입탐지 시스템의 개념과 네트워크 트래픽이 침입탐지 시스템에 미치는 영향에 대해 논의 되었다. II 장에서는 Hacking-Free-Packet 필터링 드라이버 구현을 위한 MS Windows 인터미디엇 드라이버 구조 및 작동과정에 대해 논의 하였다. IV 장에서는 Hacking-Free-Packet을 정의하고, 그에 따른 프로토콜 분석방법과 필터 방법 및 필터 설계방법에 대해 다루었다. V 장에서는 적용된 필터를 구현하고 적절한 네트워크 환경에서의 실험 및 고찰을 한 후, VI 장에서 향후 추가적으로 연구되어져야 할 분야에 대해 기술하고 결론을 맺었다.

II. 침입탐지 시스템

최근에 초고속 통신망이 각 가정에 많이 보급됨에 따라서 인터넷을 통한 네트워크의 사용이 증가하고 있는데, 네트워크 사용량이 증가함에 따라 네트워크 공격기법도 증가하고 있다. 인터넷과 관련된 수많은 어플리케이션의 발전과 더불어 그에 따르는 취약점 및 공격기법이 다양화되고 있다. 그에 따른 네트워크나 호스트로의 불법적인 침입에 대한 탐지 및 방어에 대한 연구 또한 더욱 다양화 되고 있다. 이 중 침입탐지 시스템은 불법적인 접근에 대한 탐지를 하는 시스템으로 이 분야의 가장 기본이 된다. 침입탐지 시스템 와 관련된 연구분야는 크게 침입탐지에 대한 정확한 분석 알고리즘과 그에 따른 처리 속도 향상문제, 안정적인 동작에 대한 구조 설계 분야로 나누어 질 수 있다. 이러한 과제를 해결하기 위해서는 침입탐지 시스템에 대한 이해가 선행 되어야 할 것이다.

1. 기본 개념

데이터를 보호하고 해커를 막기 위한 도구의 집합을 총칭하여 컴퓨터 보안이라고 한다. 이러한 보안을 위협하는 공격의 유형은 다음과 같다(Stallings, 1997).

- 방해(Interruption) : 시스템의 일부가 파괴되거나 사용할 수 없게 되도록 하는 것 이다.
- 가로채기(Interception) : 비인가자들의 불법적인 접근에 의한 신뢰성에 대한 공격으로, 네트워크 상에서 패킷을 가로채거나, 파일과 프로그램의 불법 복사 등을 들 수 있다.
- 불법수정(Modification) : 비인가자들의 불법적인 접근/변경에 의한 무결성에 대한 공격이다. 네트워크 상에서 패킷을 가로채기 한 후에, 패킷 내용을 수정하는 방법이다.

- 위조(Fabrication) : 네트워크 상에 위조된 패킷을 삽입하거나 파일 등에 내용을 위조하는 방법이다.
- 위장(Masquerade) : 비인가자가 인가자로 위장해서 공격하는 형태로 가장 일반적인 방법은 다른 사용자의 인증에 관한 정보를 도용하는 것이다.
- 재전송(Reply) : 적법하게 사용되는 패킷을 불법적인 공격에 이용해서 재전송하는 방법이다.
- 서비스거부(Denial Of Service) : 현재 수행되는 서비스를 정상적으로 사용되거나 관리되지 못하게 방해하는 방법이다. 이 공격에 의해서 네트워크 전체가 마비되는 경우도 발생할 수 있다.

위와 같은 보안 위협과 공격들을 방어하기 위해서는 적절한 보안 서비스가 수행되어야 한다. 보안 서비스에는 다음과 같은 것들이 있다.

- 기밀성 : 컴퓨터 시스템의 정보 및 전송 정보가 인가자만 읽을 수 있도록 해주는 서비스이다.
- 인증 : 메시지가 왔다고 주장하는 곳으로부터 전송되어 왔음을 수신자에게 확인시켜주거나, 시스템 내에 있는 자원을 접근하려는 요구에 대하여 사용자의 신분확인을 수행하는 서비스이다.

- 무결성 : 정보가 인가자에 의해서만 변경이 가능한 것이다. 즉, 허가 없이 변경이 되지 않도록 하는 서비스이다.
- 부인봉쇄: 송신자나 수신자가 전송 메시지를 부인하지 못하도록 하는 서비스이다.
- 액세스 제어 : 사용자의 식별에 관한 서비스 및 허가된 사용자가 허가된 범위에서 정보나 자원에 접근할 수 있도록 허용하는 서비스이다.
- 가용성 : 허가된 인가자가 필요로 할 때, 컴퓨터 시스템 자원을 이용할 수 있게 구현 하는 서비스이다.

보안을 위한 시스템 (침입탐지 시스템이나 방화벽)에서는 이러한 보안 서비스들을 적절하게 구현해야 한다. 예방을 하기 위해서는 송수신 되는 모든 패킷에 대한 모니터링과 분석을 통한 탐지가 이루어 져야 한다. 탐지 부분은 공격에 대한 빠르고 정확한

탐지 및 대응이 필요하게 되는데, 침입 탐지 시스템인 침입탐지 시스템 가 이용되어진다.

2. 침입탐지 시스템의 구성 요소 및 분류

침입탐지 시스템은 1980년경부터 미국에서의 연구가 시작되었는데, 침입탐지 시스템은 외부로부터의 허가되지 않은 접근, 호스트에 대한 공격행위, 해킹 및 호스트 내부 사용자의 허가되지 않은 행위 등을 감시하여 시스템 및 네트워크 운영자에게 통보하여 주고, 필요한 대응을 취하는 시스템을 의미한다.

1) 침입탐지 시스템의 구성 요소

침입탐지 시스템은 다음과 같은 3가지 중요한 구성 요소로 구성된다.

- 정보원 : 침입이 발생했는지를 판단하는 이벤트 정보는 여러가지 소스(Source)로부터 온다. 소스들은 네트워크, 호스트, 어플리케이션의 모니터링으로부터 수집된다.
- 분석 : 정보원을 분석하여 침입이 발생하고 있는지, 발생했었는지를 파악하는 부분이다. 분석 시도는 오용 탐지와 비정상상태 탐지로 나눌 수 있다.
- 응답 : 침입을 탐지하면 어떤 활동을 하게 된다. 여기에는 능동적 방법과 수동적 방법이 있다. 능동적 방법은 침입탐지 시스템이 자동적으로 시스템에 개입을 하는 것이고, 수동적 방법은 관리자에게 탐지보고를 하는 것이다.

2) 침입탐지 시스템의 분류

침입탐지 시스템은 크게 아래와 같이 분류된다.

- 분석을 기반으로 하는 분류
  - Misuse Detection

   - Anomaly Detection

- 데이터 소스를 기반으로 하는 분류
   - Host based Detection
   - Multi-host based Detection
   - Network based Detection
   - Application based Detection

- 타이밍에 따른 분류
   - Interval Detection
   - Real-Time Detection

- 응답에 따른 분류
   - Active Response System
   - Passive Response System

(1) 분석을 기반으로 하는 분류

① 오용탐지(Misuse Detection)

잘 알려진 공격의 패턴을 미리 가지고 있다가 패턴을 비교해서 공격을 파악하는 방법이다. 알려진 공격과 관련된 패턴을 시그너쳐(Signature)라고 하는데, 오용 탐지는 때때로 시그너쳐 기반의 탐지라고도 부른다. 이 방법은 패턴을 비교하여 공격을 파악하므로 공 격을 정확하게 탐지해서 매우 효과적이다. 또한 시스템 관리자가 보안에 대해서 잘 알 지 못하더라도 시스템의 보안 문제를 추적할 수 있도록 해주는 장점이 있다. 반면에, 알려진 공격만을 탐지할 수 있다는 단점이 존재하므로, 새로운 공격의 시그너쳐 들을 지속적으로 갱신 시켜야만 한다.

② 비정상상태 탐지(Anomaly Detection)

호스트나 네트워크의 비정상적인 행동을 탐지하는 방법이다. 이 탐지방법은 호스트나 사용자나 네트워크 접속의 정상 행동을 프로파일로 구축하고 있다. 이 프로파일들은 정상적인 작동을 하는 범위내의 데이터들로 구성되어 있다. 이 비정상상태 탐지에

사 용되는 방법이나 기술들은 다음과 같다.

- 문턱(Threshhold)탐지 : 사용자의 어떤 특성이나, 시스템의 행동이 수로써 표현되는 방법이다. 어떤 특성들은 주어진 시간에 사용자에 의해서 억세스 되는 파일의 수, 시스템에 접속 시도의 실패한 수, 프로세스에 의한 CPU 사용률을 포함한다.
- 통계적인 방법 : 주기적인(프로파일된 특성의 분배가 특정한 패턴에 맞는)것과 비주기적인(프로파일된 특성의 분배가 시간에서 관측되거나, 과거의 수치들의 집합으로 된)것이 속한다.
- 규칙기반 방법 : 관측된 데이터가 받아들일 수 있는 취급 패턴으로 정의되는 비주기적인 통계적인 방법과 비슷하다. 수나 양이 아니라, 규칙으로써 특별한 패턴과는 다른다.
- 신경망, 유전 알고리즘, 면역 시스템 모델을 포함하는 방법들 : 이 방법은 공격에 대한 자세한 정보가 없어도 공격을 파악할 수 있고, 오용 탐지 방법을 위해서 공격에 대한 시그너쳐를 생성할 수 있다는 장점이 있고, 사용자나 네트워크의 예측하지 못한 행동 때문에, 엄청난 수의 잘못된 탐지를 하게 될 수도 있다. 즉, 경고 오류를 남발하고, 구현 비용이 크다는 단점이 존재한다.

(2) 데이터 소스를 기반으로 하는 분류

① 호스트기반의 탐지(Host based Detection)

이 침입탐지 시스템들은 컴퓨터 시스템 내에서 정보를 수집한다. 호스트 기반의 침입탐지 시스템들은 보통 운영체제의 감사기록(audit trails)과 시스템 로그들을 정보원으로 이용 한다. 운영체제의 감사기록은 운영체제의 커널 레벨에서 항상 생성된다. 그리고 시 스템 로그들보다 자세하게 보호를 잘 받는다. 시스템 로그들은 감사기록 보다 훨씬 작고 이해하기 쉽다. 이 침입탐지 시스템들은 호스트 내부에서 이벤트를 모니터링 할 수 있기 대문에 네트워크 기반의 침입탐지 시스템에서 알 수 없는 공격을 탐지할 수 있고, 네트워크 패킷이 암호화 되더라도 동작을 수행할 수 있다. 스위치 기반의 네트 워크에 영향을 받지 않으며, 트로이 목마를 탐지할 수 잇고, 단편화를 이용한 공격을 쉽게 탐지가 가능한 장점이 있다. 그러나, 각각의 호스트들을 모니터링 하도록구

성
되고 관리되므로, 정보를 관리하는 것이 어렵고, 호스트를 공격하는 공격에 의해서 기능을 상실할 수도 있으며, 자신에게 오는 패킷만을 받으므로 네트워크 스캔을 탐지하는데 적합하지 않다. 또한, DoS(Denial-of-Service) 공격에 의해 기능을 상실할 수도 잇다. 운영체제의 audit trails를 사용하므로, 정보를 저장할 시스템의 저장공간을 요구하게 되며, 호스트의 자원을 사용하므로 호스트 성능에 영향을 미치는 단점이 존재한다.

② 다중 호스트 기반의 탐지(Multi Host based Detection)

이 시스템은 호스트 기반탐지와 비슷하나 하나의 호스트가 아닌 여러 호스트들로부터 생성되고 수집된 audit data를 침입 여부 판정에 사용하며, 여러 대의 호스트를 그 탐 지 영역으로 하기 때문에 호스트간의 통신을 통해 침입 판정에 필요한 정보를 교환하게 된다.

③ 네트워크 기반의 탐지(Network based Detection)

이 침입탐지 시스템은 네트워크 패킷을 수집하고 분석함으로써 공격을 탐지한다. 하나 의 네트워크 세그먼트 하에서 하나의 네트워크 기반의 침입탐지 시스템이 네트워크 세그먼트에 연결된 다수의 호스트들을 보호하기 위하여 모니터링 한다. 이 침입탐지 시스템은 커다란 네트워크를 소수의 침입탐지 시스템들로 모니터링할 수 있고, 침입 탐지 시스템들은 네트워크에 작은 영향만을 준다. 그리고 많은 보이지 않는 공격에 대 해 높은 보안을 유지시켜주는 장점을 가지고 있으나, 트래픽이 많은 네트워크에서는 모든 패킷을 수집하는 것이 어렵고, 사소한 패킷을 놓치게 되면, 공격을 파악하지 못 할 수도 있게 된다. 또한, 스위치 기반의 네트워크에서는 적용을 하기 힘들며, 패킷이 암호화 되면 패킷을 분석할 수 없어 공격을 탐지할 수 없게 된다. 그리고, 침입탐지 시스템은 공격이 성공했는지는 알 수 없고, 공격의 초기 단계만을 탐지할 수 있다. 또한, 몇몇 침입탐지 시스템들은 단편화(Fragment)를 이용한 공격(정, 2001. 김, 2001b)을 탐지할 수 없다는 단점이 존재한다. 그리고, 단편화를 이용한 공격을 탐지하더라도 침입탐지 시스템과 네트워크의 성능을 저하시켜 사실상, 네트워크 기반의 침입탐지 시스템에서는 탐지가 불가능하다고 말할 수 있다.

④ 응용 프로그램 기반의 탐지(Application based Detection)

이 침입탐지 시스템은 응용프로그램에서 이벤트 발생을 분석하는 호스트 기반의 침

입탐지 시스템의 특수한 형태이다. 이 침입탐지 시스템에서 사용하는 대부분의 정보원들은 응용프로그램 처리 로그 파일들이다. 이 시스템은 권한이 있는 사용자가 그들의 권한을 높이는 행위 같은 의심스러운 행동을 탐지한다. 권한이 없는 활동을 하는 사용자를 추적하는데 사용되는 응용프로그램과 사용자간의 상호작용을 모니터링 할 수 있고, 암호화된 환경에서도 사용할 수 잇다는 장점이 있다. 그러나, 응용프로그램 로그가 잘 보호되지 않음으로써 많은 취약점을 가지고 있고, 트로이 목마나 소프트웨어 조작 공격을 잘 파악하지 못하는 단점이 존재한다.

(3) 타이밍에 따른 분류

① 인터벌 기반의 탐지(Interval Detection)

이 침입탐지 시스템은 정보를 실시간으로 수집하지 않는다. 예전의 많은 호스트 기반의 침입탐지 시스템들이 이 방식을 이용하였다. 호스트 기반 침입탐지 시스템은 파일과 같은 시스템의 audit trails에 의존한다. 이 침입탐지 시스템은 능동적 응답을 수행할 수 없다.

② 실시간 탐지(Real-Time Detection)

이 침입탐지 시스템은 정보를 실시간으로 수집한다. 네트워크 기반의 침입탐지 시스템에서 이 방식을 사용한다. 실시간 침입탐지 시스템에서는 공격을 탐지하자마자 빠른 시간 안에 보고를 할 수 있어야 한다.

(4) 응답에 따른 분류

① 능동적 응답 시스템(Active Response System)

이 방법은 크게 3가지 종류로 다음과 같이 나눌 수 있다.

- 의심되는 공격에 대해서 추가적인 정보를 수집한다.
- 환경을 수정한다. 침입탐지 시스템에서는 공격을 차단하기 위해서 여러가지 방법을 쓰게된다. 첫번째, TCP의 Reset 패킷을 공격자의 접속에 끼어 넣어 접속이 끊기도록 한다. 두번째, 라우터나 방화벽을 재구성하여 공격자의 패킷을 통과시키지 않는다. 세번째, 공격자에 의하여 사용되는 서비스나 프로토콜, 네트워크 포트를 라우터나 방화벽에서 차단 시키도록 재구성한다. 네번째, 최악의 경우에는 어떤 네트워크 인터페이스에서 사용되는 모든 접속을 차단시킨다.

- 침입자에 대항하는 행동을 수행한다. 공격자의 호스트나 사이트에 대해 정보를 얻는 것을 시도하거나 공격하도록 하는 방법이다. 그러나, 공격을 하는 것은 좋은 방법이 안되고, 공격자들은 대부분 IP Spoofing 방법을 사용해서 공격을 하므로 정보를 알아내는 것은 힘들다.

② 수동적 응답 시스템(Passive Response System)

이 방법은 크게 2가지 종류로 다음과 같이 나눌 수 있다.

- 공격을 탐지한 것을 관리자에게 보고하는 방법.
- 몇몇 침입탐지 시스템은 네트워크 관리 시스템의 일부 기능으로써, 구현되는 경우가 있는데, 이러한 시스템들은 탐지를 보고하는데, 네트워크 프로토콜인 SNMP를 이용해서 보고하는 방법이다.

이러한 침입탐지 시스템은 잘 가공된 보안 관련 감사 데이터(audit data)를 요구하는데, 이러한 것을 만족시키기 위해서 보안관련 감사 데이터의 수집 기술, 추후 검사를 위한 데이터의 저장기술, 보안 관련 감사 데이터의 분석 및 해석 기술, 사용자에 대한 쉬운 인터페이스 제공 기술이 적용되어야 한다.

3. 침입탐지 절차

침입탐지 시스템이 불법적인 접근을 탐지하는 과정은 Fig. 1과 같은 단계를 가지며 크게 다음의 4 과정으로 나눠진다.

- 정보 수집 단계 (Raw Data Collection) : 이 단계는 시스템 로그를 읽어 들이거나 네트워크 패킷을 모니터링 하는 단계이다.
- 정보의 가공 및 축약 단계 (Data Reduction & Filtering) : 수집된 정보를 필터링을 하거나, 필요시 단편화 된 데이터를 제조합 하는 과정을 포함하는 과

정을 말한다.

- 침입 분석 및 탐지 단계 (Analysis & Intrusion Detection) : 가공된 데이터를 실제로 분석하고, 불법적인 접근인지를 탐지해 내는 과정이다.
- 보고 및 조치 단계 (Reporting & Response) : 불법적인 침입이라고 판명 됬을 때 해당 사항에 대해 어떠한 조치를 취해햐 하는지 결정하는 단계이다.

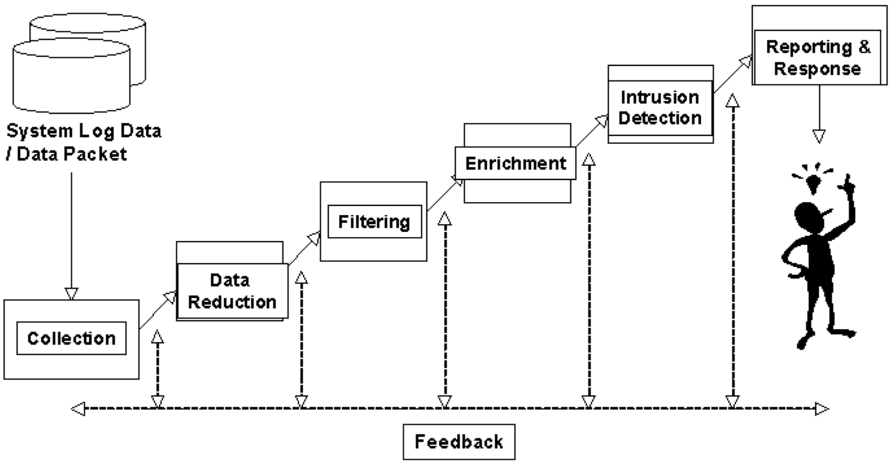

Fig. 1. Process of Intrusion Detection.

침입탐지 절차에서 이 논문이 주목하는 곳은 정보의 가공 및 축약 단계이다. 이 단계에서의 정확한 처리는 다음 단계인 침입 탐지 절차에 직접적으로 영향을 주고, 해당 침입탐지 시스템의 성능을 좌우할 수 있는 단계인 것이다.

4. 네트워크 트래픽에 대한 침입탐지 시스템

네트워크 기반 침입 탐지 시스템은 수신되는 모든 패킷을 조사하고, 때론 재구성 하여 침입을 탐지해야 한다. 하지만 그것은 현실적으로 불가능한 것으로 모든 패킷을 조사하기 위해 집중한다면 침입에 대한 실시간 분석이 어려워질 수 있기 때문이다. 여기

서 침입탐지 시스템은 딜레마에 빠지게 된다. 우선 침입탐지 시스템이 요구사항을 두 가지로 정해본다면, 모든 패킷을 모니터링해야 되는 것과 실시간으로 패킷이 침입인지를 빠르게 결정해야 하는 것 이다.

침입탐지 시스템의 성능을 향상시키기 위한 탐지 알고리즘 개선에 대한 많은 연구가 이루어 지고 있다. 보안 업체에서 만들어 지고 있는 상업용 침입탐지 시스템은 소프트웨어 기반으로 제작되어진다. 최근에는 하드웨어적인 임베디드 시스템으로 만들어 지기도 하는데, 이 두 분야의 각각의 장단점에 따라 추구하는 바가 틀리나 기본적인 요구사항은 동일하다. 그것은 실시간으로 거대한 네트워크 트래픽 하에서 침입을 탐지 하려 하는 것 이다. 본 논문에서도 이러한 기본적인 요구사항을 해치지 않으면서 극복할 수 있는 시스템을 제안하는 것이다. 추가적으로 현재 시스템에 구성된 침입탐지 시스템에의 영향을 최소화 하기 위해 소프트웨어적으로 해결해 나가는 것이다.

Mirecom Lab 의 보고서에 의하면 기가비트 이더넷에서 986.94 Mbps 백그라운드 트래픽을 갖는 네트워크에서 침입탐지 비율은 44% 정도라고 한다(Mire, 2001). 물론 이 트래픽이 한계적인 트래픽이라고 할지라도, 트래픽이 증가하면 침입탐지 시스템의 성능은 현저하게 떨어짐을 알 수 있다. 또한 Charles 와 Andrew는 패스트 이더넷 환경에서 백그라운드 트래픽이 40% 정도 였을 때 침입 탐지율은 60%정도라고 보고하고 있다(Iheagwara, 2002). 침입탐지 시스템이 하드웨어(캐시 사이즈, CPU 속도, 메모리 크기, I/O 속도)에 의존적이긴 하지만, 이를 무시하고라도, 소프트웨어적으로 패턴 매칭 알고리즘이나 세션 추적 알고리즘은 어느 정도 한계가 있음을 알 수 있다. 즉, 이것은 침입탐지 시스템이 네트워크상의 모든 패킷을 캡쳐할 수 없다는 것을 보여주는 예가 될것이다.

침입탐지 시스템에서의 패킷 손실이 일어나는 이유는 크게 네트워크 손실과 시스템 손실로 볼 수 있을 것 이다. 여기서 말하는 네트워크 손실은 패킷 충돌에 의한 트래픽이 발생한 스위치나 버퍼 오버플로어에 의한 허브나 라우터에서의 손실을 말한다. 네트워크 손실은 피할 수 없을 뿐더러 복구도 되지않는다. 침입탐지 시스템 입장에서 본다면 이러한 손실은 네트워크 안의 어떠한 시스템에도 해를 끼치지 않을 것이므로 고려할 필요가 없다. 즉 어차피 손실된 패킷이므로 해킹에 사용될 수도 없다는 것이다.

시스템 손실은 침입탐지 시스템의 커다란 문제점에 의해 발생한다. 그것은 침입탐지 시스템이 사용하는 패킷 캡쳐 모듈의 문제점일 것이다. 대부분의 상업용 침입탐지 시

스템은 pcap 이라는 패킷 캡쳐 모듈을 사용한다. 이 pcap이라는 모듈은 프로토콜 드라이버단에서 제작된 패킷 드라이버이다. 이에 대해서는 다음 장에서 자세히 설명할 것이다. 침입을 탐지하기 위해 침입탐지 시스템은 네트워크 카드로부터 이 드라이버간에 메모리 복사 과정을 수행할 것이다. 이것은 네트워크 카드로 패킷이 수신되는 속도가 메모리 복사 속도 보다 빠른 경우 발생할 수 있을 것이다. 이는 네트워크 대역폭이 커지고, 이에따른 네트워크 트래픽이 증가함으로서 더욱 빈번히 일어날 것이다.

본 논문은 시스템 손실의 원인인 패킷 캡쳐 모듈의 한계를 극복하는 것이다. 이러한 한계를 극복하기 위해 하드웨어적으로 구성하고자 하는 연구도 있지만, 자원 효율 적인 면에서 소프트웨어 적으로 구현하고자 하는 것이다. 또한 대부분의 침입탐지 시스템 사용하는 패킷 캡쳐 모듈이 다음 장에서 설명할 프로토콜 드라이버 계층에서 구현되어 있음에 따라, 다양한 패킷 캡쳐 모듈에 독립적으로 접목될 수 있는 시스템을 제안하고자 한다.

III. MS Windows 인터미디엇 드라이버

MS(Microsoft)는 Windows 시스템에서의 커널 모드 네트워크 드라이버로 미니포트 드라이버, 인터미디엇 드라이버, 프로토콜 드라이버를 제공한다. 미니포트 드라이버는 NIC(network interface card)을 직접적으로 제어하고 상위 레벨 드라이버와의 인터페이스를 제공하는 드라이버를 말하는 것으로 일반적으로 제공되어지는 네트워크 어뎁터 드라이버이다.

프로토콜 드라이버는 TCP/IP 와 같은 프로토콜 드라이버로 어플리케이션에서 만들어진 데이터에 계층간 프로토콜 헤더를 추가해서 미니포트 드라이버로 전송하거나, 미니포트 드라이버에서 수신된 이더넷 프레임에서 프로토콜 헤더를 제거하여 상위 레벨로 전송하는 등의 기능 및 프로토콜 스텍을 구현하는 드라이버이다.

인터미디엇 드라이버는 상위 레벨의 프로토콜 드라이버와 하위 레벨의 미니포트 드라이버 사이에 위치하며, 프로토콜 드라이버에 알려지지 않은 미디어에 대한 변환을 수행하기 위해 제공된 기술이다. MS는 개발자가 인터미디엇 드라이버를 디자인하고 추가함으로써 미디어 변환 이외에 많은 기능을 추가할 수 있도록 하였다.

1. MS Windows 네트워크 드라이버

MS Windows 플랫폼 환경에서 네트워크 관련 하드웨어를 추상화 하기 위해서 NDIS라는 라이브러리가 사용되어진다. 이 라이브러리는 Fig. 2에서 보듯이 가장 하위 계층의 실제 하드웨어를 다루는 드라이버에서부터 가장 상위 계층의 네트워크 전송 계층까지의 모든 네트워크 드라이버 사이의 인터페이스를 제공하며, 또한 드라이버간의 상태정보나 각종 파라미터(함수 포인터, 핸들 등)에 대한 관리를 한다.

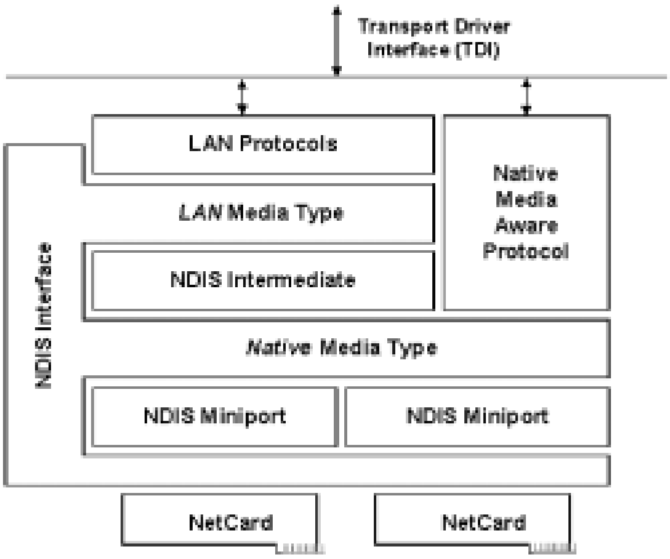

Fig. 2. Windows Network driver structure.

그림에서 보듯이, 가장 하위계층에는 실제 물리적인 장치인 네트워크 카드가 위하며, 그 바로 위에는 네트워크 카드를 인터페이스 하는 미니포트 드라이버가 위치하게 된다. 커널모드 중 가장 상위에는 프로토콜 드라이버가 위치하며, 이 드라이버는 TDI(transport driver interface)와의 통신과 하위 계층과의 통신을 인터페이스 하게 된다. 이러한 프로토콜 드라이버에는 TCP/IP, IPX/SPX, NETBEUI 등이 있으며, 침입탐지 시스템에서 패킷 캡쳐를 위해 사용되어지는 모듈 드라이버도 이 계층에서 구현되어 진다. 프로토콜 드라이버와 미니포트 드라이버가 다루는 미디어 타입은 서로 다를 수가 있는데, 이를 변환하는 중계역할을 할 수 있도록 인터미디엇 드라이버가 두 드라이버 사이에 계층화 되어질 수 있다. 이러한 Windows의 네트워크 구조에 기반하여, 제안된 패킷 필터 모듈은 침입탐지 시스템이 사용하는 패킷 캡쳐 모듈보다 하위계층인 인터미디엇 드라이버로 구현되어졌다.

2. 인터미디엇 드라이버의 구조 및 동작 과정

1) 인터미디엇 드라이버

Fig. 3은 인터미디엇 드라이버가 포함되어 있지 않았을 때의, 즉 일반적인 Windows 네트워크 드라이버 구조이다.

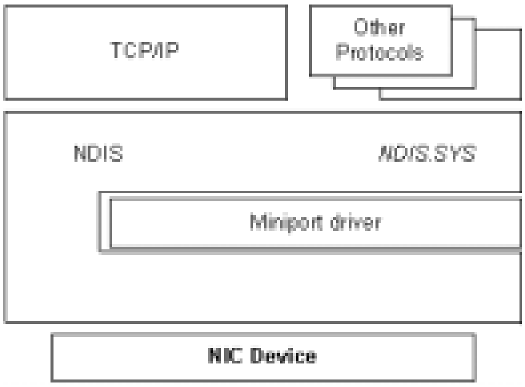

Fig. 3. General Windows network driver structure.

NIC 장치와 미니포트 드라이버, 프로토콜 드라이버가 계층화 된 것을 볼 수 있으며, NDIS는 이 세 계층에 대한 인터페이스를 제공한다. 즉, 프로토콜 드라이버의 함수들(ProtocolXXXX)은 NDIS 함수를 호출하게 되고, 이 호출 받은 NDIS 함수는 미니포트 드라이버의 함수(MiniportXXXX)를 호출하게 된다. 그리고, 미니포트 드라이버 함수는 NDIS 함수를 통해 실제 NIC 장치를 제어할 수 있게 된다.

NDIS는 앞에서도 설명했듯이 이러한 함수에 대한 지원뿐 아니라 호출관계 및 연결 인자등과 같은 명세서를 제공하게 된다. 이러한 일반적인 Windows 네트워크 구조에 인터미디엇 드라이버를 계층화해서 추가할 수 있는데, 이는 LAN 매체가 아닌 것에 대한 미니포트 드라이버와 프로토콜 드라이버 사이에서의 에뮬레이터 역할을 수행한다. 데이터 송신시 인터미디엇 드라이버는 상위계층에서 LAN 포맷인 미디어를 받고 그 미디어들을 NIC 고유의 미디어(예: PPPOE, ATM)로 바꾼다. 그리고 그것들을 NIC에 대한 미니포트 드라이버에게 보낸다. 수신시에는, 인터미디엇 드라이버 하위의 NIC 드

라이버로부터 지시된 미디어를 LAN 호환 포맷으로 바꾸고 그 변환된 미디어를 상위 계층으로 올려 보낼 수 있다. 이러한 인터미디엇 드라이버의 활용도는 4가지로 분류될 수 있다.

- 미니포트 드라이버와 프로토콜 드라이버 사이에서의 미디어(매체) 변환
- 보안이나 그밖의 용도로의 데이터 필터링
- 로드 밸런싱
- 네트워크 데이터의 모니터링 및 수집

Fig. 4는 Windows 기본 네트워크 드라이버 구조 사이에 인터미디엇 드라이버가 계층화 되었을 때의 모습이다. 그림에서 보듯이 인터미디엇 드라이버는 프로토콜 드라이버와 미니포트 드라이버 사이에 계층화 됨을 알 수 있다.

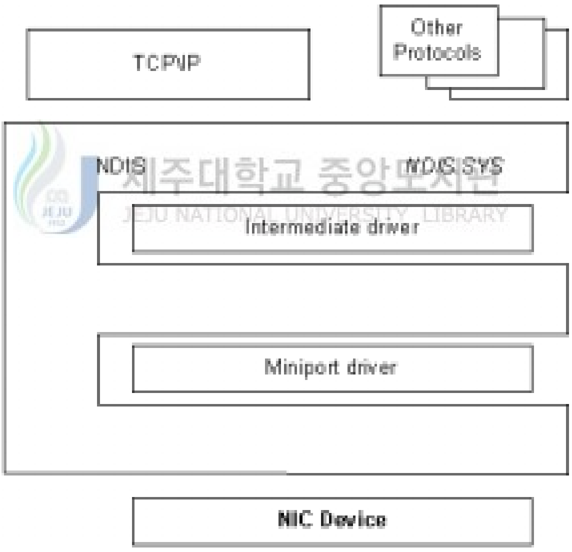

Fig. 4. Intermediate driver in the NDIS.

Fig. 5는 인터미디엇 드라이버의 내부구조를 보여주는 것으로, 내부적으로 Upper-Edge 는 미니포트 드라이버 인터페이스를 가지고 있고, Lower-Edge는 프로토콜 드라이버 인터페이스를 가지며, 둘 사이는 내부적으로 바인딩 되어져 있다

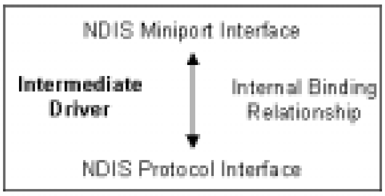

Fig. 5. Internal binding in intermediate driver.

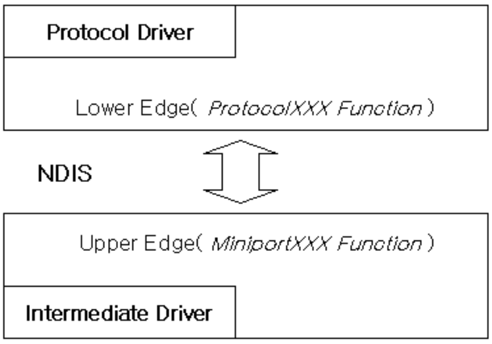
Protocol Driver
Lower Edge(ProtocolXXX Function)
NDIS
Upper Edge(MiniportXXX Function)
Intermediate Driver

Fig. 6. Binding protocol driver to intermediate driver.

Fig. 6에서 보듯이 프로토콜 드라이버의 Lower-Edge 함수는 인터미디엇 드라이버의 Upper-Edge 함수와 바인딩 되어진다. 즉, 프로토콜 드라이버는 자기와 미니포트 드라이버가 바인딩 된 것으로 여기게 되는 것이다. 마찬가지로 아래 그림 7처럼 미니포트 드라이버의 Upper-Edge는 인터미디엇 드라이버의 Lower-Edge 와 바인딩 되어져, 상위 계층에 프로토콜 드라이버가 있는 것으로 인식하여 동작하게 된다.

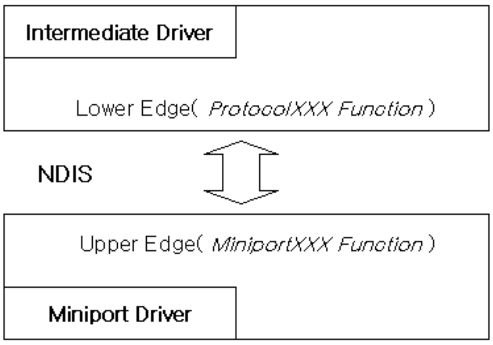
Intermediate Driver
Lower Edge(ProtocolXXX Function)
NDIS
Upper Edge(MiniportXXX Function)
Miniport Driver

Fig. 7. Binding protocol driver to intermediate driver.

이러한 인터미디엇 드라이버 모델을 디자인 함으로서 네트워크로 송/수신 되는 모든 데이터는 이 계층을 거치게 되는데, 인터미디엇 드라이버 디자인에는 아래의 4가지 과정이 고려되어야 한다.

- 인터미디엇 드라이버 초기화/종료
- 인터미디엇 드라이버의 패킷 관리
- 인터미디엇 드라이버의 송/수신
- 인터미디엇 드라이버의 셋/쿼리

2) 인터미디엇 드라이버 초기화

인터미디엇 드라이버의 초기화 과정은 Fig. 8에서 보듯이 크게 인터미디엇 드라이버를 등록 과정과 상위 프로토콜 드라이버와 하위 미니포트 드라이버와의 바인딩 과정으로 이루어진다.

윈도우에서 드라이버의 초기화 루틴의 시작 지점은 DirverEntry 이다. DriverEntry 에서는 인터미디엇 드라이버의 Lower-Edge 함수와 Upper-Edge 함수를 등록하고, 거기서 얻어지는 NdisAdapterHandle 과 NdisProtocolHandle을 연계시킨다. 이는 앞에서 설명한 인터미디엇 드라이버 내의 프로토콜 인터페이스와 미니포트 인터페이스의 내부 바인딩 과정이다.

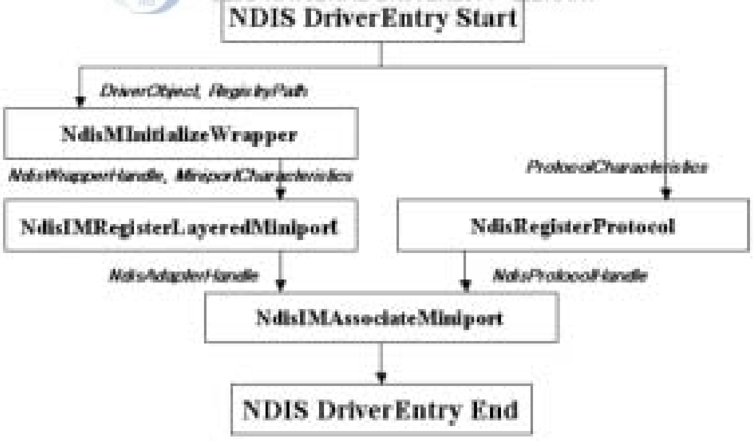

Fig. 8. Initialize intermediate driver.

이후에 인터미디엇 드라이버는 상위 레벨의 프로토콜 드라이버 와 하위 레벨의 미니포트 드라이버와의 바인딩 과정이 시작된다.

Fig. 9는 그러한 바인딩 과정이 설명되어진 것이다. 이는 인터미디엇 드라이버 내의 ProtocolBindAdapter가 호출 되면서 시작된다. 이는 NIC이 시스템에 설치되면 NDIS에 의해 호출되는 함수로 내부적으로 두 가지 과정을 거치게 된다. 우선 NDIS 함수인 NdisOpenAdapter를 호출함으로써 미니포트 드라이버를 얻게된다. 이 과정을 통해 미니포트 드라이버의 BindingHandle 및 미디어등을 알게된다. 여기서 얻어진 BindingHandle은 미니포트 드라이버 마다 고유한 것으로 다수의 NIC 이 설치되어 있을 때는 각각의 BindingHandle을 얻을 수 있고, 이를 통해 각각의 장치를 제어할 수 있게 된다.

그 후에 ProtocolBindAdapter는 인터미디엇 드라이버를 프로토콜 드라이버로서 등록하기 위해 NdisIMInitializeDeviceInstanceEx 함수를 호출하게 되고, 이 NDIS 함수는 인터미디엇 드라이버의 MiniportInitialize를 호출하게 된다. 이는 다시 NDIS 함수인 NdisMSetAttributeEx를 호출함으로서 인터미디엇 드라이버를 미니포트 드라이버로서 등록하게 된다. 이 과정에서 MiniportHandle을 얻게 되는데, 이는 데이터 수신시 상위레벨에 전송하기 위해 사용되어지는 핸들이다.

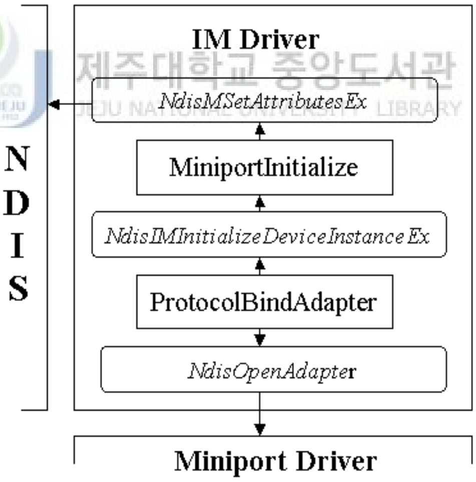

Fig. 9. Binding intermediate driver.

3) 인터미디엇 드라이버의 패킷 관리

NDIS 구조에서 데이터의 이동을 위해 NDIS_PACKET 이라는 데이터가 사용되어진

다. 이 NDIS_PACKET을 이용해 송/수신 되는 데이터의 포인터를 상위 드라이버나 하위 드라이버로 보내게 된다. NDIS_PACKET은 단지 NDIS_BUFFER를 가리키는 포인터로, 실제 데이터를 이동시키는 대신 이 주소 값을 이동 시킴으로써 손쉬운 송/수신 과정을 하게 된다. 다시 NDIS_BUFFER는 가상 메모리의 번지 값을 지시하고, 이 가상 메모리의 번지 값은 실제 데이터를 가리키게 된다. 인터미디엇 드라이버를 초기화 하는 과정에서 이 NDIS_PACKET과 NDIS_BUFFER에 대한 초기화 과정을 수행하여야 한다. 또한 이를 사용하기 위해서는 미리 Pool 이라는 저장공간을 확보해야 하는데 NDIS_PACKET이 사용되어지는 Pool은 NdisAllocatePacketPool이라는 NDIS 함수에 의해 초기화 되어지고, NDIS_BUFFER가 사용되는 Pool은 NdisAllocateBufferPool에 만들어진다. 이러한 공간이 확보가 된 후에 송/수신 과정에서 데이터 전송에 필요한 NDIS_PACKET을 NdisAllocateMemory와 같은 함수로 할당하여 사용하게 된다. Fig. 10은 NDIS_PACKET과 NDIS_BUFFER, 메모리와의 관계를 그림으로 도식화 한 것이다.

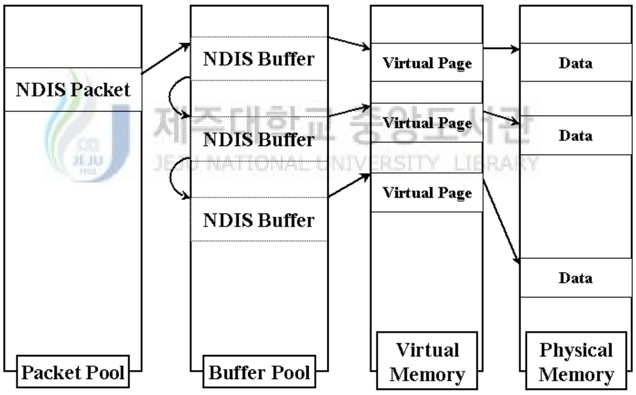

Fig. 10. NDIS_PACKET.

4) 인터미디엇 드라이버의 송/수신

(1) 송신 과정

인터미디엇 드라이버의 송신과정을 알아보기 이전에, 우선 인터미디엇 드라이버가 계층화 되지 않았을 때의 일반적인 구조에서의 프로토콜 드라이버 와 미니포트 드라이버 간의 송신과정을 이해할 필요가 있다. Fig. 11은 일반적인 네트워크 구조에서의

송신과정을 보여준다. 프로토콜 드라이버에서 데이터를 전송하기 위해 NdisSend라는 NDIS 함수를 호출하게 되고, 이 NdisSend는 미니포트 드라이버의 MiniportSend 함수를 호출하게 된다.

NIC 을 통해 데이터가 전송되면 송신 완료라는 인터럽트가 발생하게 되고 이는 NdisMsendComplete 함수를 호출함으로써 프로토콜 드라이버의 ProtocolSendComplet를 발생시켜 송신 과정을 종료 시킨다.

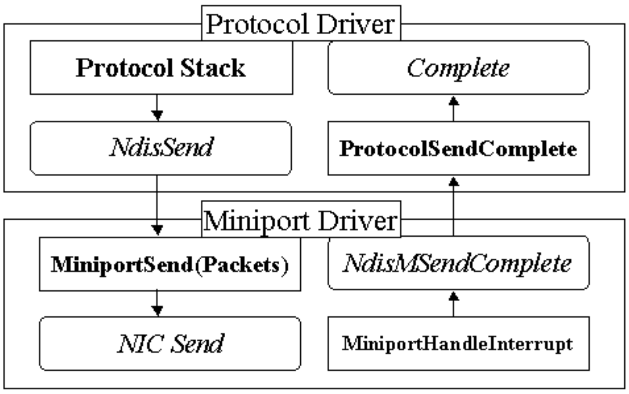

Fig. 11. General send routine.

여기에 인터미디엇 드라이버가 계층화 되면 프로토콜 드라이버에서 호출된 NdisSend 는 미니포트 드라이버의 Upper-Edge 함수가 아닌 인터미디엇 드라이버의 Upper-Edge 함수인 MiniportSend을 호출하게 된다. 그럼 사용자는 이 Upper-Edge 함수에서 NdisSend를 다시 호출 시킴으로써 데이터가 미니포트로 전달되도록 한다. 송신 완료에 대한 관계에도 인터미디엇 드라이버를 거쳐 상위 계층으로 전달된다. Fig. 12는 인터미디엇 드라이버가 계층화 되었을 때의 송신과정을 보여준다.

이는 앞에서 설명했듯이 인터미디엇 드라이버가 프로토콜 드라이버와 미니포트 드라이버 사이의 교량 역할을 하는 것을 보여준다. 즉, 모든 송신 패킷은 사용자가 만든 I인터미디엇 드라이버를 거쳐야만 전송 되어질 수 있는 것이다. 추가적으로 NdisSend 인자 값으로는 NDIS_PACKET과 BindingHandle이 사용되어진다.

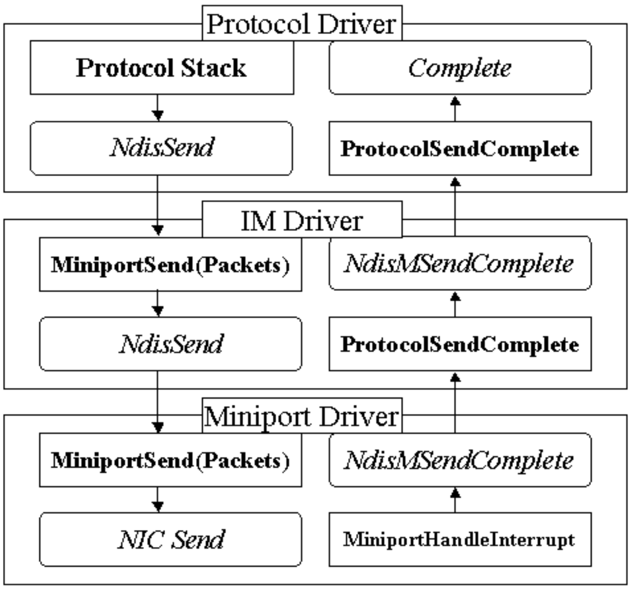

Fig. 12. Send routine in intermediate driver.

(2) 수신과정

수신 과정도 송신과정이랑 거의 유사한 과정을 띈다. Fig. 13에서 보듯이, NIC을 통해 데이터가 들어오면 NdisMIndicateReceivePacket 함수가 호출되어 지고, 이는 인터미디엇 드라이버의 Lower-Edge 함수인 ProtocolReceive함수를 호출하게 된다. 사용자는 내부코드에서 N disMIndicateReceivePacket을 호출함으로써 상위 프로토콜 드라이버단의 Protocol Receive을 호출하여 수신할 수 있다. 수신이 정상적으로 되면 프로토콜 드라이버에서 NdisReturnPacket이 호출되고 이는 인터미디엇 드라이버의 Upper-Edge 함수인 Mi niportReturnPacket을 통해 미니포트 드라이버의 MiniportReturnPacket을 호출하게 된다. 이렇게 수신이 완료되면 NIC에서는 해당 리소스를 지우게 된다.

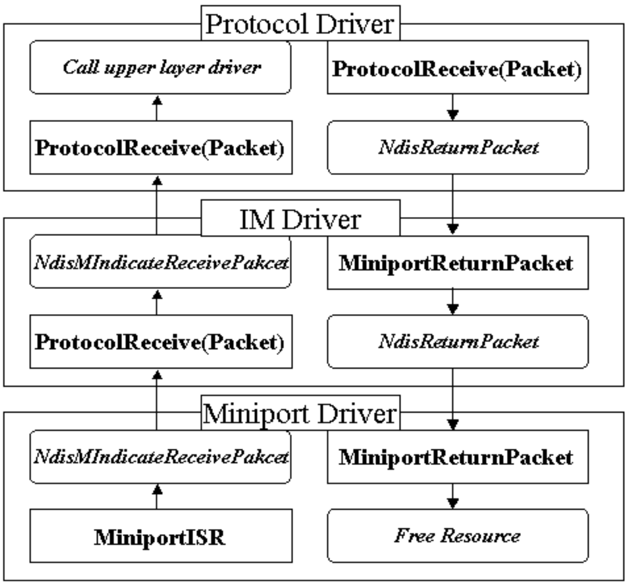

Fig. 13. Receive routine in intermediate driver.

5) 인터미디엇 드라이버의 셋/쿼리

인터미디엇 드라이버가 초기화가 된 이후에, 인터미디엇 드라이버는 하위의 NIC 드라이버의 동작 특성들을 쿼리 하거나 내부 상태를 설정하고, 하위 계층의 NIC 드라이버와 함께 바인딩에 대한 인자들을 설정할 수 있다. Fig. 14에서 처럼 인터미디엇 드라이버는 MiniportQueryInformation 와 MiniportSetInformation 함수에서 프로토콜 드라이버로부터 쿼리와 셋 요구를 받을 수 있다. 쿼리/셋에 대한 완료 루틴은 Fig. 15와 같이 ProtocolRequestComplete 함수를 통해 이루어 진다.

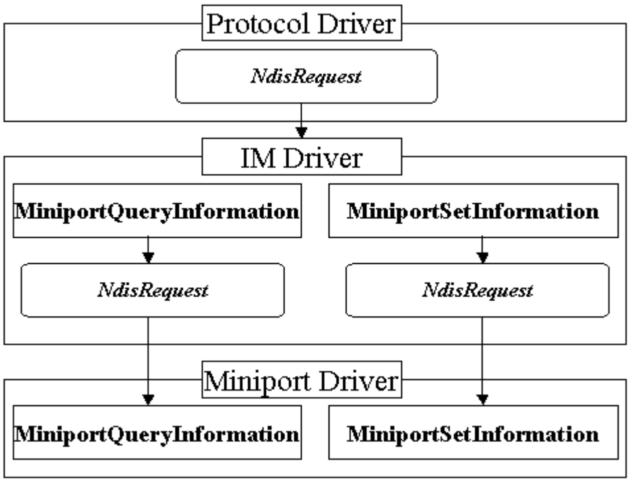

Fig. 14. Set/Query in intermediate driver.

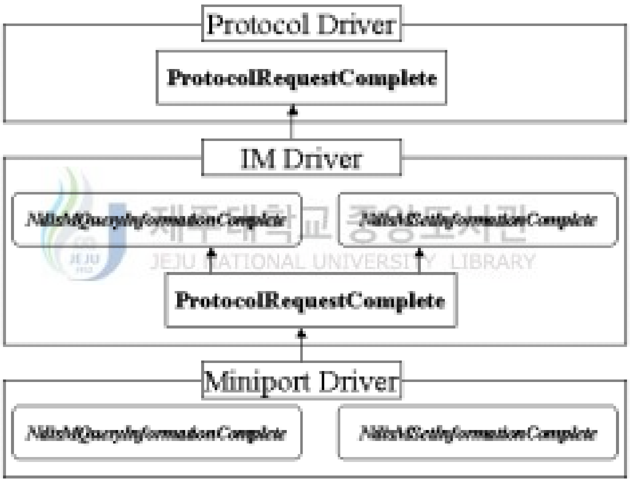
Protocol Driver
ProtocolRequestComplete
IM Driver
NdisIMQueryInformationComplete
NdisIMSetInformationComplete
ProtocolRequestComplete
Miniport Driver
NdisIMQueryInformationComplete
NdisIMSetInformationComplete

Fig. 15. Set/Query completion in intermediate driver.

6) 인터미디엇 드라이버의 종료

인터미디엇 드라이버는 NIC이 더 이상 이용될 수 없을 때 NDIS가 호출하는 프로토콜UnbindAdapter로부터 NdisCloseAdapter를 호출함으로써 밑의 NIC 드라이버로부터 바인딩을 해제 시킨다. 예를 들면, 만약 NIC이 타임아웃 이고 NDIS가 기능장애를 갖는 NIC에 대해서 NIC 드라이버의 MiniportHalt함수를 호출했다면, NDIS는 인터미

디엇 드라이버의 ProtocolUnbindAdaprer 함수를 호출할 것이다. Fig. 16은 종료시 사용되는 함수 호출 관계를 보여준다.

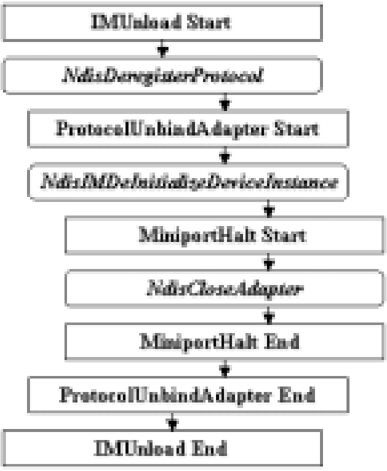

Fig. 16. End routine of intermediate driver.

3. 인터미디엇 드라이버에서의 패킷 캡쳐 및 분석

이 장에서는 MS Windows 의 네트워크 구조 및 인터미디엇 드라이버에 대해 살펴보았다. 이러한 인터미디엇 드라이버의 이러한 계층화 구조로 인해 (프로토콜 드라이버와 미니포트 드라이버 사이의 계층화) 인터미디엇 드라이버를 제작함으로서 패킷을 가로채거나, 분석 할 수 있게 된다. 즉, 네트워크 패킷은 항상 인터미디엇 드라이버를 통과하므로, 해당 루틴 (MiniportSend, ProtocolReceive) 내부에 필터링 알고리즘을 집어넣을 수 있다.

IDS인 경우, 수신된 네트워크 패킷에 대한 분석을 요구하므로, 인터미디엇 드라이버의 수신과정을 자세히 살펴볼 필요가 있다(Fig. 17). 네트워크 패킷이 수신되면, 미니포트 드라이버에서 NdisMIndicateReceivePacket 이라는 NDIS 함수를 호출하게 되고, 이 함수는 인터미디엇 의 Lower-Edge함수인 ProtcolReceivePacket 을 호출하게 된다. 이때 NDIS_PACKET과 수신된 미니포트 드라이버 핸들을 넘겨 주게 된다.

ProtocolReceivePacket 단에서는 상위 프로토콜 드라이버로 데이터를 넘겨주기 위해 NdisMIndicateReceivePacket을 호출하게 된다. 프로토콜 드라이버에서는 데이터가 수신되면, NdisReturnPacket을 호출하여 수신 완료를 알리고, 이를 받은 인터미디엇 드라이버의 MiniportReturnPacket에서도 미니포트 드라이버로 NdisReturnPacket을 호출하여, 그 상태를 알리게 된다. 이렇게 패킷 수신이 완료 되면, 미니포트 드라이버는 할당되었던 자원을 제거하게 된다.

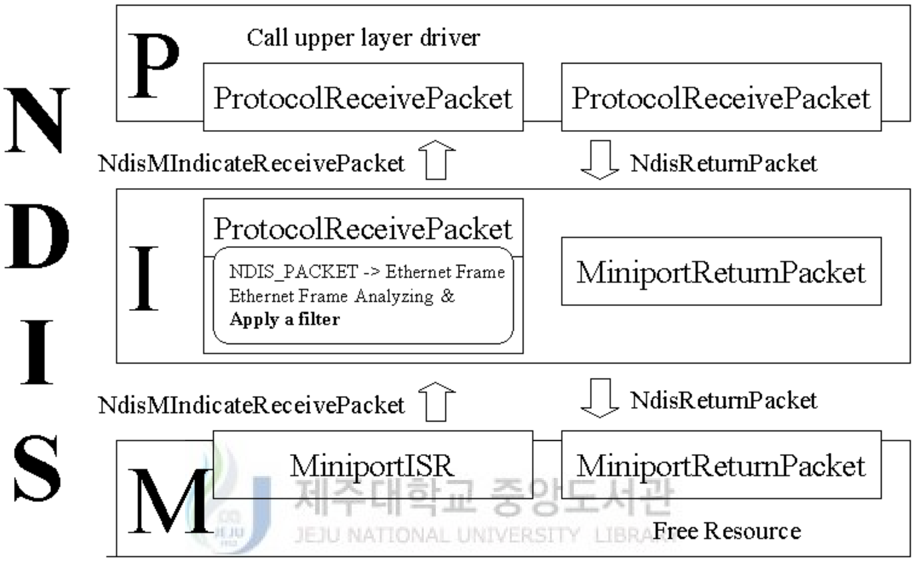

Fig. 17. Packet capture & analyze in receive routine, intermediate driver.

수신된 패킷을 분석하기 위해, 인터미디엇 드라이버의 ProtocolReceivePacket 내부에 필터 모듈을 추가하였다. 수신된 패킷은 NDIS_PACKET 형태를 가지므로, 우선 NDIS_PACKET에서 이더넷 프레임을 추출하였고, 추출된 이더넷 프레임을 분석하고 필터 모듈을 적용하였다.

IV. Hacking-Free-Packet 필터

침입탐지 시스템은 호스트로 들어오는 모든 패킷을 조사해야 할 필요성이 있다. 이는 당연한 것으로 언제, 어떠한 불법적인 접근이 있을지 모르기 때문이다. 하지만 네트워크 대역폭이 기가 비트급으로 증가하고 있는 현실에서 이러한 모든 패킷에 대한 분석은 침입탐지 시스템의 부하를 증가시킬 것이다.

이에 대해 완벽하게 무해한 패킷을 가려내어 침입탐지 시스템이 이 패킷을 분석하지 않도록 한다면 빠른 침입탐지 능력 향상에 많은 도움을 줄 것이다. 이러한 작업을 더욱 효율적으로 하기위해 패킷 필터 모듈이 우선 개발 되어져야 한다.

1. 패킷 필터링

패킷 필터링이란 특정한 패킷에 대하여 일정한 정책을 적용하여 그 패킷을 받아들일 것인지 말 것인지(accept or reject)를 결정하는 것을 말한다.

일반적으로 네트워크 인터페이스 카드는 자신에게 오거나 브로드캐스트 되는 패킷만을 시스템으로 올려준다. 수신된 패킷은 프로토콜 드라이버에서 프로토콜 스택에 의해 처리된다. 하지만 네트워크의 모든 패킷을 모니터링하기 위해서는 네트워크 인터페이스 카드를 PROMISCUOS 모드로 열어서 모든 패킷을 수신할 수 있도록 해야한다. 이러한 패킷 필터는 다음의 조건을 만족하여야 한다(Guang, 1998).

- Real-time performance : 패킷 필터는 데이터 링크 계층으로부터 빠르게 원시(Raw) 패킷을 캡쳐해야 하며, 짧은 시간안에 그것을 처리해야 한다.
- No Packet Dropping : 드랍되는 패킷이 있어서는 안된다. 이것은 침입탐지 시스템에서 아주 중요한 문제이다. 드랍된 패킷에 있는 정보로 인해 침입탐지에 대한 중요한 정보를 놓칠 수 있기 때문이다.

- Flexibility : 다른 프로토콜에 적용하기 쉽도록 수정이 용이해야 한다.
- Scalability : 침입탐지 시스템의 성능 저하 없이 새로운 침입 패턴을 추가할 수 있어야 한다.

1) BPF (berkley packet filter)

데이터 링크 계층에서 원시(Raw) 패킷을 수집하기 위해 사용되는 필터 시스템에는 BPF(berkley packet filter) 필터가 주로 사용되어진다. BPF 필터는 공개용 패킷 필터 시스템으로 UNIX나 Windows 시스템 모두에서 응용되어지고, 사용되어진다. BPF 는 크게 네트워크 구조적인 면과 패킷 필터 두 부분으로 나누어 지는데, 네트워크 구조적인 면은 NIC 에서 원시(Raw) 패킷을 받아오거나, 사용자 프로세스로 넘겨주는 구조에 대해 정의한다. 패킷 필터 부분은 Fig. 18 에서 보듯이 NIC과 사용자 프로세스 사이에 위치하며, 패킷을 분석하여 필터 여부를 결정하는 부분이다. (McCanne, 1993)

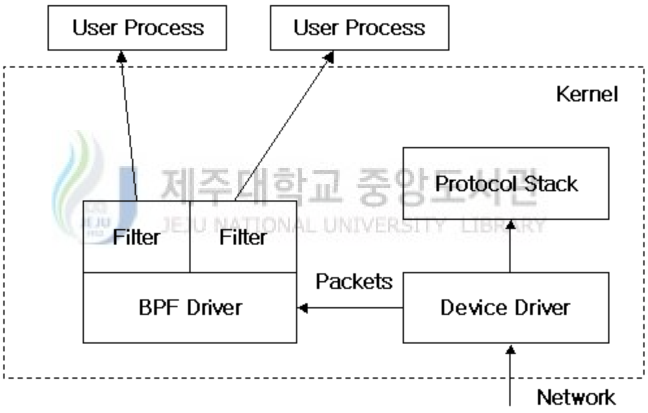

Fig. 18. BPF system architecture.

여기서 패킷 필터 부분은 CFG(control flow graph)를 바탕으로 구현되어졌는데, 이는 다른 패킷 필터 시스템(NIT, DLPI)에서의 트리 구조 시스템과 차별되어지는 기능이다.

트리 모델은 스텐포드에서 1980년에 개발되어 지속적으로 많이 사용이 되고 있다. 이것은 패킷을 필터링할 때 허용 여부(accept와 reject)를 판단하기 위하여 AND와 OR로 연결된 트리를 순회한다. 아래의 Fig. 19는 수신된 패킷에서 호스트가 foo인 패킷

을 찾아내는 필터를 트리 모델로 나타낸 그림이다.

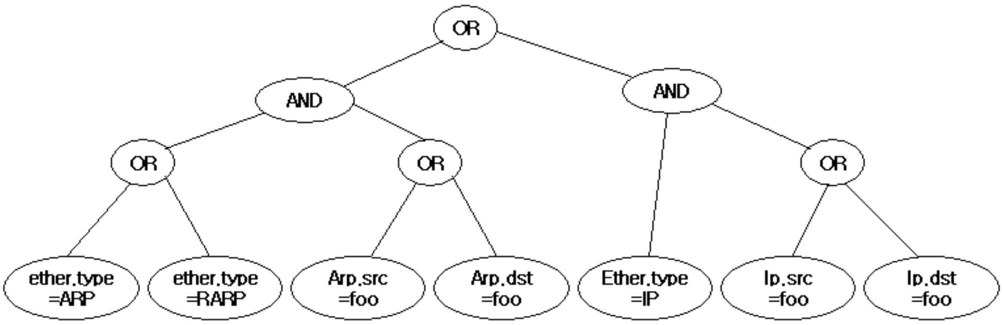

Fig. 19. Tree filter function for “host foo”.

이에반해 CFG는 각각의 비교를 노드로 하여 참이면 오른쪽, 거짓이면 왼쪽의 노드로 옮겨가서 패킷을 받아들일지를 결정하는 방법이다. Fig. 20은 위에서와 같이 패킷에서 foo 호스트를 찾아내는 예를 보여주는 것이다.

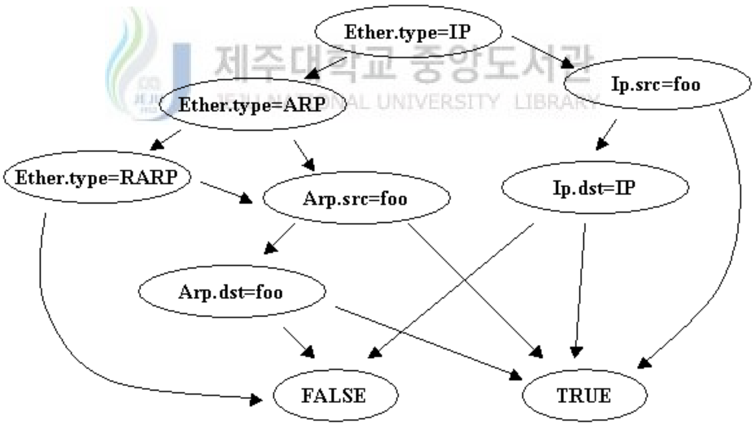

Fig. 20. CFG filter function for “host foo”.

두개의 필터를 비교해 봤을 때, BPF 모델(CFG)이 트리 모델보다 처리과정이 짧다는 것을 알 수 있는데, 예를 들면 트리 모델에서는 “ether.type == IP” 인 경우에도 “ether.type == ARP”를 처리해야만 패킷의 필터링 유무를 결정할 수 있지만, CFG 모

델에서는 중복적인 검사를 피할 수 있다.

BPF 필터의 각 경로는 특정 패킷에서 비교할 필요가 있는 정보에 대한 표현이다. 즉, 패킷 정보는 필터에 기억되어지고, 한번 계산되어진 값은 매번 다시 계산되어질 필요가 없게 된다.

2. ASL에 의한 패킷 필터 모델 구현

인터미디엇 드라이버 단에서 Hacking-Free-Packet 필터링 시스템으로 위에서 언급한 CFG 모델에 기반한 필터 시스템을 제안하고자 한다. 이 필터는 기본적으로 침입 패턴과 그에 대한 응답으로 표현 되어지는 것으로, 룰 기반 기술 언어인 ASL(Audit Specification Language)을 이용하여 정의되어진다(Guang, 1998).

Pattern | *condition -> reaction*

ASL은 위와 같은 형식으로 표현되어진다. *Pattern*은 이 Rule에 사용되어지는 값을 말한다. 패킷 필터 시스템에서는 수신되는 네트워크 패킷, 즉 이더넷 프레임을 말한다. *Condition*은 패킷에 대한 간단한 비교 과정으로, 패킷의 각각의 필드 값에 대한 연산 과정 이다. *Reaction*은 *Condition* 이후에 행해져야 할 행동을 말한다. 이를 통해 필터링 조건을 표현하면, CFG 형식으로의 변환이 가능해 진다.

1) 패킷 접근 방법

일반적으로 패킷의 헤더를 각각의 헤더 구조체에 넣어 사용하는 것은 패킷의 구조를 설명하는데 유용하다. 하지만 여기서는 패킷의 각각의 필드에 대한 정보가 필요하지 않으므로, 패킷을 바이트 흐름으로 다룬다. 예를 들면 이더넷 헤더의 프로토콜 필드는 “*(short)packet[12]*”로 나타내어 질 수 있다. 이러한 접근의 단점은 헤더각각의 필드 정보내용을 다 알 수 없고, 사용되어질 때마다 타입 변환을 해주어야 한다는 것이다. 즉, 필요한 값에 접근하여, 그 값을 사용하고자 할 때, 해당 필드의 크기만큼 short(2Byte)나 int(4Byte)등으로의 변환이 필요하다. 하지만, 불필요한 필드의 접근을 하지 않는 다는 장점이 있다.

또한 여기서는 각각의 헤더의 정보를 알기 위해 헤더의 연결을 이용한다. IP 헤더는 이더넷 헤더의 프로토콜 필드가 IP 인 경우의 이더넷 헤더의 데이터 영역이며, 유사하게 TCP 헤더는 IP 헤더에 연결되어진다. 예를 들어 TCP 헤더의 출발지 포트 필드는 IP 헤더의 옵션부분이 없는 경우 "*(short)packet[34]*"로 나타내어질 수 있다. 이러한 방식은 필터를 유연하고, 확장성을 가지도로 한다. 즉, 헤더구조를 정의하지 않아도, UDP나 ICMP 와 같은 프로토콜에 대해 쉽게 지원할 수 있다. 또한 다른 프로토콜에도 쉽게 접목되어 질 수 있도록 한다.

위와 같은 방식을 사용하기 위해서는 패킷의 정확성이 요구되어진다. 예를 들어 TCP의 출발지 포트를 접근하기 위해서는 먼저 해당 패킷이 IP 패킷인지, 또한 TCP 패킷인지 확인하는 과정들이 선행되어져야 한다.

2) ASL을 이용한 예

ASL의 사용법을 설명하기 위해서 간단한 예와 함께 설명해 보겠다. 만일 수신된 패킷중 foo라는 호스트에서 온 패킷을 찾아내기 위한 형식은 아래와 같다. (foo 호스트의 IP 주소가 11.22.33.44인 경우)

Packet(p)|(p.s_addr == 11.22.33.44) -> message("host foo")

*Packet*은 데이터 계층에서 수신된 이더넷 프레임을 말한다. 조건부는 간단히 "*p.s_addr*"에 대한 간단한 BOOLEAN 비교 과정이다. 이는 패킷 *p*의 IP 헤더의 출발지 주소이다. 앞에서 설명했듯이 이러한 조건을 위해서는 먼저 패킷 *p* (이더넷 프레임)이 IP 프로토콜인지 확인해야 되는 과정이 필요하다. ASL에서의 연속적인 조건문의 추가는 AND를 이용해 추가되어 진다. 그러므로 위 식은 아래와 같이 표현되어져야 한다.

Packet(p)|(p.e_type == ETHER_IP) && (p.s_addr == 11.22.33.44) -> message("host foo")

위와 같이 표현된 ASL 식은 BPF의 Fig. 21에서처럼 CFG 모델로 표현할 수 있다.

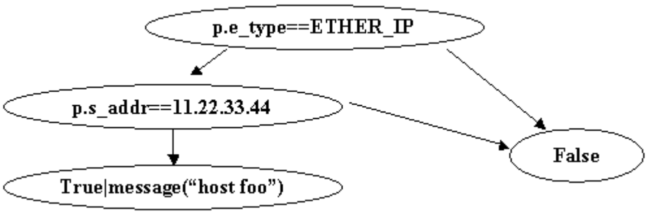

Fig. 21. Sample filter for "host foo".

이러한 ASL을 이용한 CFG는 전체적인 필터 구조를 간략화 할 수도 있는데, 예를 들어 IP 헤더의 프로토콜 필드에 TCP,UDP,ICMP, IGMP가 아닌 값에 대한 필터를 ASL 식으로 나타내면, 아래와 같이 나타낼 수 있을 것이다.

Packet(p)| (p.e_type == ETHER_IP) && (p.protocol != IP_TCP)
&& (p.protocol != IP_UDP) && (p.protocol != IP_ICMP)
&& (p.protocol != IP_IGMP) -> message("IP Unknown protocol")

이를 일반적인 CFG 모델로 표현한다면 Fig. 22와 같을 것이다.

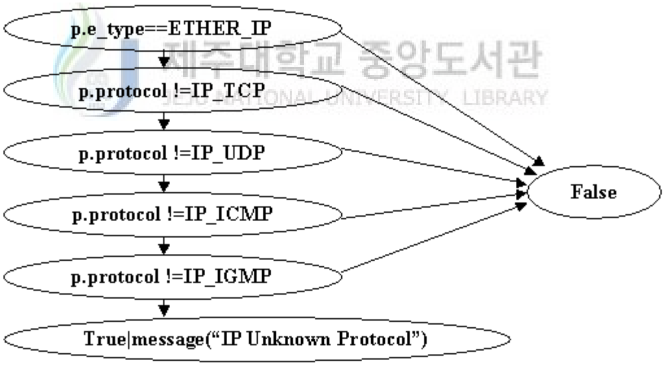

Fig. 22. Sample filter for "IP Unknown Protocol".

위 구조는 같은 값에 대한 중복된 비교문을 노드로 표시함으로써 더욱 최적화 시킬 수 있다. 이러한 목적은 필터 트리 크기를 줄여줄 수 있다. Fig. 23은 Fig. 22를 더욱 최적화 시킨 결과이다.

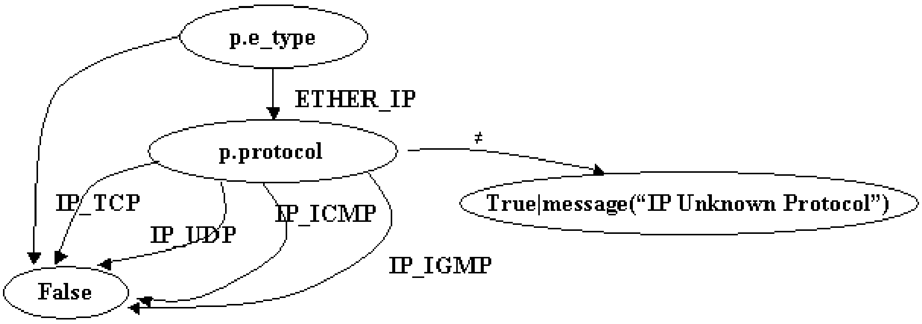

Fig. 23. Improved filter for "IP Unknown Protocol".

3) OR 연산

ASL에서는 조건문의 추가 시 AND를 사용한다. 하지만 때에 따라서, 조건문에 OR 연산이 필요한 경우가 있을 수 있다. 예를 들어 네트워크 기반 침입탐지 시스템에서, 수신된 패킷에 호스트가 foo인 경우를 찾기 위해서는 p.tcp_saddr 이나 p.tcp_daddr 두 부분에 대한 조사가 필요하다. 그 경우의 ASL 식은 아래와 같을 것이다.(호스트 foo의 IP가 11.22.33.44인 경우)

Packet(p)|(p.etype == ETHER_IP) && ((p.s_addr == 11.22.33.44)
||(p.d_addr == 11.22.33.44)) -> message("host foo")

OR 연산인 경우 앞부분 조건문의 *reaction*은 원 식의 *reaction*과 동일하고 FALSE로의 노드는 없다. FALSE로의 노드 대신 다음의 조건문을 수행하게 한다. 위 식에 대해 CFG를 모델링 해보면 Fig. 24와 같다.

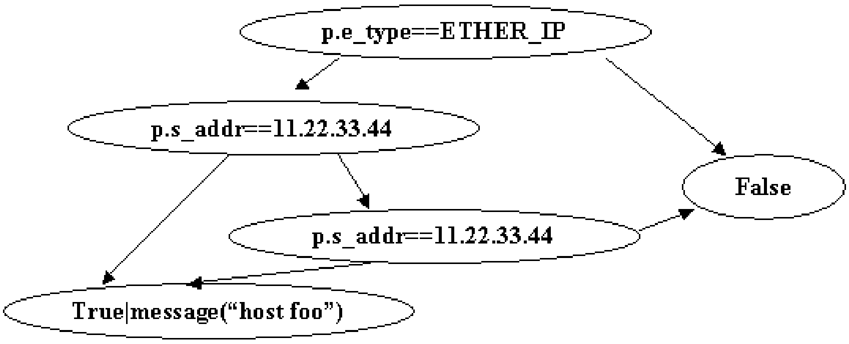

Fig. 24. Sample for OR filter for "host foo".

이러한 ASL에 의한 패턴 매치 방법은 패킷의 각각의 헤더 필드를 연속적인 데이터 흐름으로서 독립적으로 다루는데 있고, 이를 통해 필터 모델을 쉽게 CFG로 최적화 시킬수 있는 방법을 제공한다.

3. Hacking-Free-Packet

현재 인터넷 환경에서 HTTP(Web)이 커다란 영역을 가지고 있고, 이에 따른 많은 다양한 멀티미디어 서비스 또한 증가 추세에 있다. 이러한 멀티미디어에 대한 수요가 늘어나고 있고, 이러한 멀티미디어 서비스는 네트워크 대역폭의 많은 부분을 차지하는 현실이고, 앞으로 더욱 늘어날 것이다(Thompson, 1997). 이러한 패킷은 대부분 무해한 패킷일 경우가 많은데 이러한 패킷은 사실상 침입탐지 시스템의 작업 능률을 떨어뜨릴 뿐이다. 또한 TCP를 통한 HTTP 패킷의 교환시 연결설정을 제외하고는 일반적인 HTTP 데이터 송수신 패킷 또한 무해한 경우가 많다. 이에 이렇게 무해한 패킷, 즉 Hacking-Free-Packet을 정의하고 필터링 할 필요가 있다(Moon, 2001).

1) HTTP에서의 Hacking-Free-Packet

만일 패킷이 정상적이고, 어떠한 해로운 정보를 가지고 있지 않다면 침입탐지 시스템이 해당 패킷을 검색하는 것은 무의미한 일이며, 단지 탐지 성능만을 떨어뜨릴 뿐이다. 인터넷에서 웹이 급속도로 성장함에 따라 다양한 멀티미디어 서비스가 제공되고 있다. 이러한 멀티미디어 콘텐츠는 자주 네트워크 트래픽을 유발 시킨다. 멀티미디어 트래픽은 침입탐지 시스템에게도 부하를 주며, 침입탐지 시스템의 성능을 떨어뜨리는 주된 요인이 된다. 그러나, 대부분의 멀티미디어 트래픽은 Hacking-Free-Packet인 경우가 대부분이다. 물론 콘텐츠를 제공하는 서버가 해킹에 피해를 입지 않았을 때를 전제로 한다. 침입탐지 시스템은 대부분의 이러한 멀티미디어 데이터는 탐지할 필요가 없을 것 이다. 예를들어, 클라이언트가 네트워크를 통해 음악을 들으려고 한다면, 우선 음악파일을 다운 받기 위해 서버와의 연결을 시도한다. 이 연결설정을 위해 최초 몇개의 패킷을 주고 받게 된다. TCP인 경우 “3-Handshaking” 과정일 것이다. 연결설정

이 이루어 지면 그 후에 음악파일이 전송될 것이다. 이 경우, 해킹에 안전한 서버로부터 전송되는 음악 파일은 시스템에 해로운 정보를 포함하고 있지 않을 것이다. 그러나, 연결 설정을 위한 패킷과 ACK 패킷(수신되는 콘텐츠 패킷에 대한 응답 패킷)은 해킹에 사용되어지거나, 다른 공격 메커니즘에 이용될 확률이 있다.

이더넷 프레임의 각각의 헤더에는 해당 패킷이 멀티미디어 스트림 패킷이라는 정보를 가지고 있지 않다. 즉, 이것은 하나의 패킷만을 가지고 이것이 어떠한 패킷인지 확인할 수 가 없음을 보여준다. 이를 위해서는 세션 재구성 기술이 필요할 것이다. 그러나 세션 재구성은 침입탐지 시스템에 오버헤드를 주므로 전체적인 탐지 성능을 감소시킬 것이다.

세션을 재구성 하는 대신, 침입탐지 시스템은 각각의 패킷의 헤더를 분석하고, 해로운 패킷인지에 대한 유무를 결정하는 것이 더 좋을 것이다. 공인된 웹 서버에서 제공하는 데이터는 대부분 해킹에 무해한 내용의 데이타일 것이다. 그러므로, 웹 서버에서 송신되는 해킹에 무해한 패킷을 찾아내어 침입탐지 시스템 앞 단에서 필터링 하는 것은 성능향상을 기대할 수 있을 것이다. Fig. 24는 웹 서버와 클라이언트의 웹 브라우저와의 연결과정을 설명한다(Stevens, 1999).

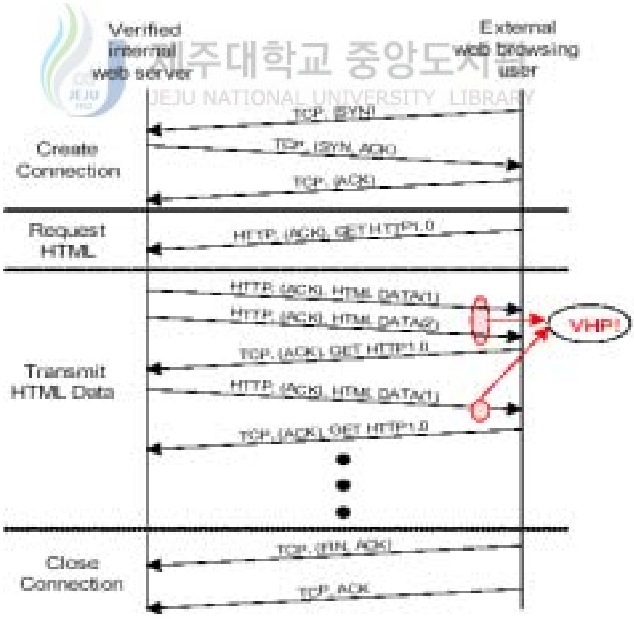

Fig. 24. The sequence of HTTP and Hacking-Free-Packet.

HTTP 데이터 송수신 중 다음 두종류의 패킷은 Hacking-Free-Packet이 될 수 없다.

(1) TCP 연결 설정 패킷

대부분의 침입탐지 시스템은 출발지 IP주소와 목적지 IP주소에 기반한 TCP 세션을 관리하며, 이 세션 정보에 의해 침입을 탐지한다. TCP 연결 설정이 없다면 침입탐지 시스템은 세션을 재구성 할 수 없다. 그리고, 해킹(공격) 기술 중에는 "SYN Flooding"이나 "ACK Storm"을 이용한 DDOS(distributed denial of service)와 같은 것이 있다. 그러므로 이 패킷은 침입탐지 시스템으로 전송되어 분석되어져야 한다.

(2) 클라이언트에서 보내어 지는 HTTP 패킷

웹서버의 대부분의 공격 당하기 쉬운점은 CGI와 같은 구조적인 약점에 기인한다. 이 경우에 해커가 쿠키와 같은 정보를 이용하여 웹 프로그램에 불법적인 실행 명령이나 스크립트를 덮어씌울 수 있다

HTTP 프로토콜의 결함을 이용한 많은 해킹이 발생하고 있는데, 대부분 HTTP 연결설정과 관련되어 있다. 공개 침입탐지 시스템인 Snort 경우 HTTP와 관련된 약 440개의 룰을 가지고 있다(Snort). 이 룰에 따르면, 잘못된 URL이나 "403 Forbidden"에 관련된 3가지 룰만이 존재하는데, 이것은 웹 서버에서 송신되는 패킷에 대한 것이고, 나머지는 외부로 송신되는 패킷에 대한 것이다. 이것은 전체 룰의 0.68%밖에 되지 않으므로 세션 패킷을 제외한 나가는 HTTP 패킷은 분석되지 않아도 될 것이고, 이로 인해 침입탐지 시스템의 침입탐지 성능은 향상될 것이다.

2) HTTP Hacking-Free-Packet 필터

위에서의 HTTP 관련 패킷중 TCP 연결설정 패킷과 클라이언트에서 나가는 HTTP 데이터 패킷을 제외한 패킷은 침입탐지 시스템에게 보내어 지지 않도록 인터미디엇 드라이버 단에서 차단 시킬 필요가 있다. 이러한 패킷을 구분해 내기 위해서는 아래와 같은 정보가 필요하다.

- TCP 패킷
- Port 정보 : 출발지 포트, 목적지 포트

- TCP 플래그 : ACK, ACKPSH
- TCP 데이터 : HTTP 데이터중 클라이언트에서 나가는 패킷(GET HTTP1.0)이 아닌 데이터

이를 ASL을 이용하여 아래와 같이 표현할 수 있다.

Packet(p)| (p.e_type == ETHER_IP) && (p.protocol == IP_TCP)
&& ((p.tcp_sport == HTTP) || (p.tcp_dport == HTTP))
&& ((p.tcp_flag == ACK) || (p.tcp_flag == ACKPSH))
&& (p.tcp_data != GET) -> drop

Fig. 25는 위 ASL 표현식에 대한 HTTP Hacking-Free-Packet 필터에의 CFG 모델이다.

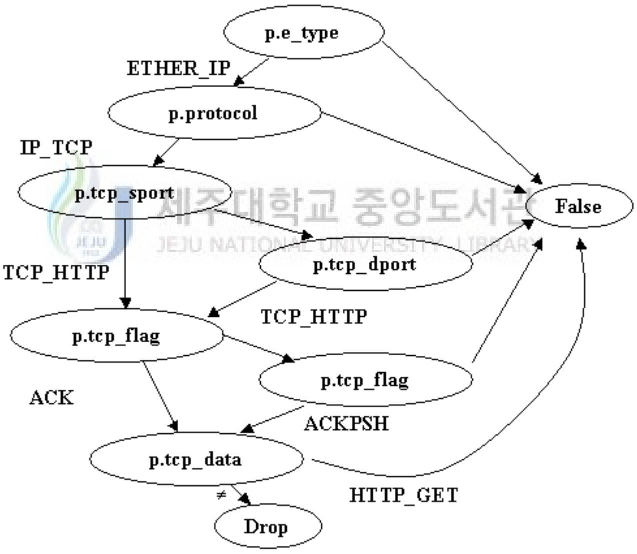

Fig. 25. HTTP Hacking-Free-Packet filter for “http packet”.

HTTP에서의 Hacking-Free-Packet Filter의 조건식은 7개가 필요함을 알 수 있다. 이 7개의 BOOLEAN 조건식 함수만을 구현하여 인터미디엇 드라이버에 필터 모듈로서 적용시킬 수 있으며, 추가적으로 NDIS_PACKET해석 루틴이 필요하다.

V. 제안된 시스템의 구현 및 고찰

인터미디엇 Hacking-Free-Packet 필터가 기존에 설치된 침입탐지 시스템에 미치는 영향을 정확하게 판단하기 위해, 다른 네트워크 트래픽의 영향을 없애도록 독립적인 네트워크 망을 구성하였으며, 기존의 침입탐지시스템과 인터미디엇 Hacking-Free-Packet 필터가 설치된 침입탐지 시스템에서의 탐지율을 측정하였다.

1. 시스템 구현을 위한 네트워크 환경

침입탐지 시스템의 탐지율을 측정하기 위해서 Fig. 26과 같은 네트워크 및 시스템을 구축 하였다.

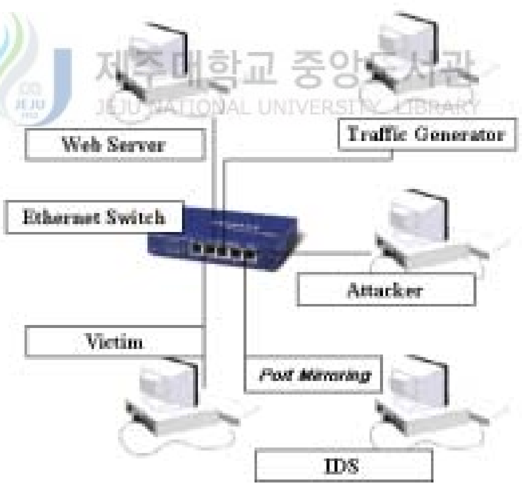

Fig. 26 The empirical network

1) 침입탐지 시스템 (IDS)

침입탐지 시스템으로는 공개용 침입탐지 프로그램인 Snort-1.9.0 Win32 버전을 사

용하였다(Snort). Snort는 네트워크 패킷을 수집하기 위해 PCAP 라이브러리를 사용하는데, 여기서는 Win32 용, Winpcap-2.3 버전을 이용하였으며, Pentium III 800Mhz 의 Windows 2000 시스템에 구축하였다(Winpcap).

2) 침입시스템 (Attacker)

침입시스템으로는 Land Attack이 사용되었다. Land Attack은 목적지 IP와 소스 IP를 같게 하여 패킷을 보내는 것으로 DOS 공격 툴의 일종이다(Insecure). 이 Land Attack 공격패킷을 다량으로 보냈을 때 침입탐지 시스템의 탐지해 내는 횟수를 탐지율로 계산하였다. Land Attack이 사용된 시스템은 Pentium Celeron 400Mhz의 Linux 커널 2.4 가 설치된 시스템이다.

3) 트래픽 발생기 (Traffic Generator)

네트워크 트래픽을 발생시키는데 두 종류의 패킷을 사용하였다. 첫째는 HTTP 패킷으로 http_load라는 프로그램을 사용하였다(ACME). http_load는 HTTP 트래픽을 무작위로 발생시키는 프로그램으로, 원하는 만큼의 트래픽을 만들어 줄 수 있었다. 두번째는 UDP 트래픽이다. 이를 위해서는 MGEM이라는 UDP 트래픽 발생 프로그램을 사용하였다(Naval). 이 시스템은 Pentium Celeron 400Mhz, Linux 커널 2.4 시스템에서 사용하였다.

4) 웹서버 (Web Server)

웹서버는 Apache를 사용하였으며, Pentium Celeron 400Mhz의 Linux 커널 2.4 시스템에 구축하였다(Apache).

5) 클라이언트 (Victim)

침입을 받을 시스템은 Pentium III 733Mhz, Windows 2000 시스템을 사용하였다.

6) 스위칭 허브 (Ethernet Switch)

위 5개의 시스템은 Extreme사의 Summit24 Switching HUB를 통해 연결되어지도록 했다. Dummy 허브가 아니므로 침입탐지 시스템으로 모든 네트워크 패킷이 보내어 질

수 있도록, 포트 미러링 설정을 하였다.

2. 인터미디엇 Hacking-Free-Packet 필터 드라이버 구현 구현

- OS : Microsoft Windows 2000 Service Pack 3
- NIC : 3Com 905B-TX
- Language : C
- Develop Environment : Microsoft Visual C++ 6]

인터미디엇 드라이버의 전체적은 골격은 MS DDK(device driver kit)에서 제공하는 샘플 소스인 PassThru를 참조하였다. NIC을 통해 네트워크 패킷이 수신되면 미니포트 드라이버 에서는 NdisMIndicateReceivePacket을 호출하게 되고, 이를 받은 NDIS는 넘겨 받은 정보를 가지고 인터미디엇 드라이버의 Lower-Edge 함수인 ProtocolReceive를 호출하게 된다. NdisMIndicateReceivePacket은 패킷이 어느 NIC에서 수신되어졌는지에 대한 정보를 가진 미니포트 핸들과 수신된 패킷의 포인터, 패킷 수를 넘겨주게 된다.

```
VOID
  NdisMIndicateReceivePacket(
    IN NDIS_HANDLE    MiniportAdapterHandle,
    IN PPNDIS_PACKET    ReceivePackets,
    IN UINT    NumberOfPackets
    );
```

인터미디엇 드라이버의 ProtocolReceive에서는 수신된 패킷을 분석(필터링)하기 위해 NDIS_PACKET을 이더넷 프레임으로의 변환과정이 추가 되어야 한다. 이를 위해서 이더넷 프레임에 대한 구조체와 변환 함수를 아래와 같이 정의했다.

```
typedef struct _ETHERNETFRAME
{
   UINT            Length;
   UCHAR           Buffer[1520];
}ETHERNETFRAME, *PETHERNETFRAME;

PETHERNETFRAME
HFPTranslateNdisPacketToEthernetFrame(IN PNDIS_PACKET pNdisPacket)
```

Hacking-Free-Packet 필터의 각 조건 함수와 노드 값은 아래와 같이 구현하였다.

```
#define ETHERNET_PROTOCOL_IP0x0008
#define IP_ PROTOCOL _TCP        0x06
#define TCP_ PROTOCOL _HTTP      0x5000
#define TCPF_ACK                 0x10
#define TCPF_ACKPSH              0x18
```

```
BOOLEAN
Ethernet ProtocolTypeIP(IN PETHERNETFRAME pEthernetFrame);

BOOLEAN
IpProtocolTCP(IN PETHERNETFRAME pEthernetFrame);

BOOLEAN
SourcePortHTTP(IN PETHERNETFRAME pEthernetFrame);

BOOLEAN
DestinationPortHTTP(IN PETHERNETFRAME pEthernetFrame);
```

```
BOOLEAN
TcpFlagACK(IN PETHERNETFRAME pEthernetFrame);

BOOLEAN
TcpFlagACKPSH(IN PETHERNETFRAME pEthernetFrame);

BOOLEAN
HttpDataNotGET(IN PETHERNETFRAME pEthernetFrame);
```

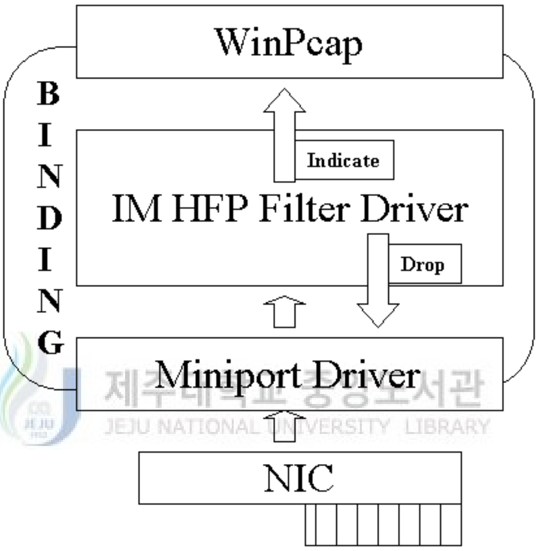

Fig. 27. Intermediate Hacking-Free-Packet filter driver.

Fig. 27처럼, 인터미디엇 Hacking-Free-Packet 필터 드라이버 를 Winpcap과 바인딩 되도록 설치하였고, 이를 통해 미니포트 드라이버에서 수신된 패킷에 대해 필터링을 적용하도록 했으며, 이를 통해 수신된 패킷에 대한 필터 여부를 결정하였다.

3. 구현 결과 및 고찰

네트워크 트래픽은 HTTP 패킷과 UDP 패킷을 발생 시키면서, 총 네트워크 트래픽을10% 에서 99% 까지 조절하였다. Land Attack 공격 패킷은 10000개를 사용했으며, 침입탐지율은 10번에 걸쳐 조사한 후 평균값을 택하였다. 네트워크 트래픽에 HTTP 패킷 비율에 따른 탐지율을 알아내기 위해 HTTP 패킷과 UDP 패킷의 비율을 변화시켜 보았다.

1) 인터미디엇 Hacking-Free-Packet 필터 드라이버의 패킷 처리 속도

침입탐지율을 측정하기에 앞서 인터미디엇 Hacking-Free-Packet 필터 드라이버의 동작속도를 측정해 보았다. 우선, 수신된 패킷은 드랍되지 않고 모두 처리가 되어, 상위 프로토콜 드라이버로 전송되는 것을 확인하였다. 수신된 하나의 패킷에 대하여 인터미디엇 Hacking-Free-Packet 필터 드라이버에 의한 처리 속도는 5000개의 패킷에 대해 조사해 보았다. Fig. 28에서 보듯이 패킷 하나당 처리속도는 평균 24.064usec가 소요되었다. 이 속도는 NIC이 수신할 수 있는 최대 대역폭인 100Mbps를 처리 가능한 속도이다. 처리속도가 떨어지는 경우도 볼 수 있는데, 이는 Windows 시스템이 기본적으로 멀티테스크 시스템으로 다른 프로세스(Snort 등)에게도 필요에 따라 자원을 할당함으로서 생기는 결과로 볼 수 있다.

2) 제안된 시스템의 침입탐지 이득 효과

Table 1은 네트워크 트래픽에 따른 침입 탐지율을 보여준다. 네트워크 트래픽은 전체 대역폭의 비율로, 10% 에서 99% 까지 조사하였다. HxUy는 네트워크 트래픽에서의 HTTP 트래픽과 UDP 트래픽의 비율을 나타내었다. '*'은 제안된 필터 드라이버가 설치된 침입탐지 시스템에서의 탐지율을 나타낸다. 우선, 결과에서 보듯이, 네트워크 트래픽이 증가할수록 침입탐지율은 감소하는 것을 볼 수 있다. 하지만, 네트워크 트래픽중 HTTP 패킷(Hacking Free Packets)의 비중이 높아질수록, 제안된 시스템이 기존의 시스템에서 보다 탐지율이 높게 측정됨을 알 수 있었다.

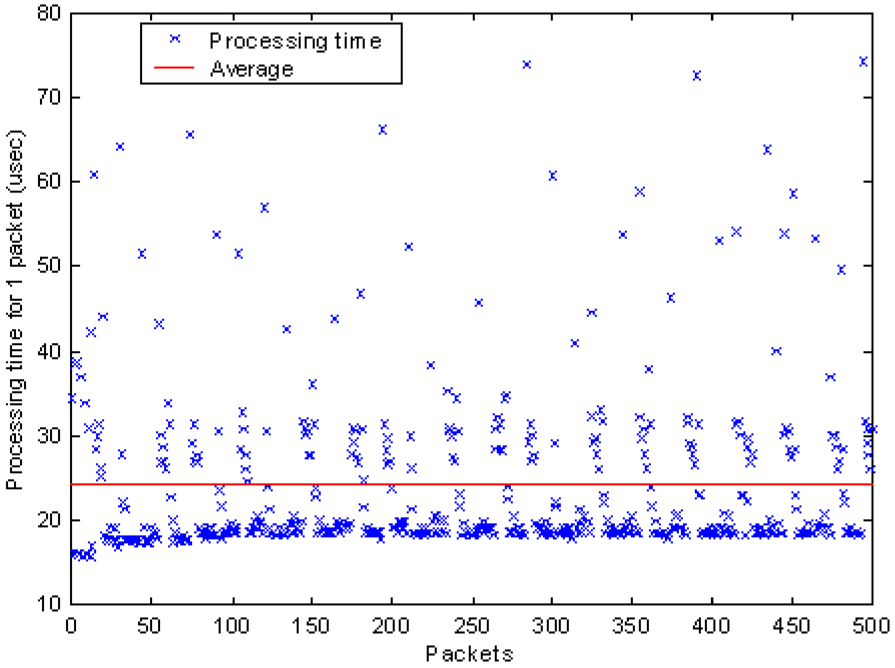

Fig. 28. Processing time for 1 packet in proposed filter

Table 1. Attack detection rate

Traffic	10%	20%	30%	40%	50%	60%	70%	80%	90%	99%
H1U9	100	100	97.5	96.9	94.2	91.8	80.9	70.4	68.5	64.2
H1U9*	100	100	100	96.4	94.2	92.2	81.2	72.2	70.3	66.2
H3U7	100	99	97.9	97	94	91.8	82.3	71.3	68.2	65
H3U7*	100	100	100	100	96.4	94.2	86.2	79.9	76.2	74.4
H5U5	100	100	97.0	96.7	94.4	91.1	82.6	71.7	68.9	64.8
H5U5*	100	100	100	100	99	97.4	88	84.2	83	81.4
H7U3	100	100	97.9	96.4	94.8	91.9	81.6	72.4	69.6	65.4
H7U3*	100	100	100	100	100	98.3	98.2	98.4	97.8	96.4
H9U1	100	100	97.9	96.8	94.1	91.2	79.9	71.6	69	63.9
H9U1*	100	100	100	100	100	100	100	99	98.4	98.2

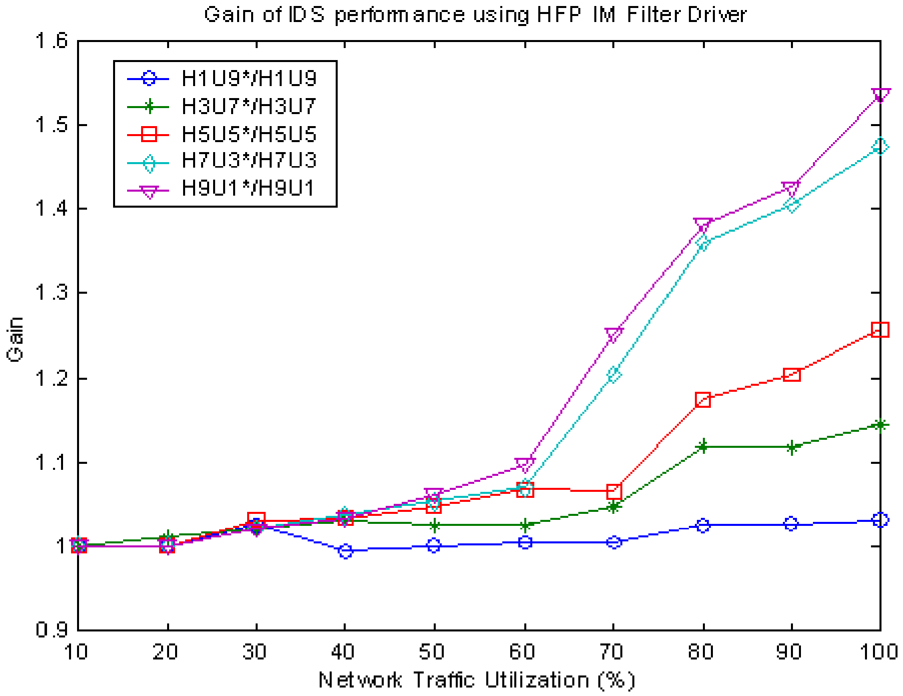

Fig. 28 Gain of IDS performance using intermediate Hacking-Free-Packet filter driver.

Fig. 28은 Table 1의 결과에 따른, 기존의 시스템에 대한 제안된 시스템의 탐지 이득을 보여준다. 그림에서 보듯이 네트워크 트래픽 사용량이 60%이상이 되었을 경우, 그 트래픽중, HTTP 트래픽이 UDP 트래픽에 비해 많은 비중을 차지할수록 탐지 이득이 현저히 높음을 알 수 있다. 하지만, 제안된 시스템은 HTTP 트래픽이 네트워크 트래픽의 적은 부분을 차지하는 경우 그다지 성능 향상을 기대 할 수 없음을 알 수 있다.

VI. 결론

인터넷 사용인구의 확산으로, 네트워크 트래픽도 폭발적으로 증가하고 있다. 침입탐지 시스템은 이러한 네트워크 트래픽을 모두 처리해야 할 필요성이 있으며, 이는 곧 트래픽 처리능력은 침입탐지 시스템의 탐지 능력과 직결된다고 볼 수 있다. 기존의 많은 침입탐지 시스템은 기가 비트급의 높은 네트워크 트래픽 상황에서 많은 한계점, 즉 침입탐지율이 급격히 떨어지는 현상을 보이고 있다.

이에 본 논문에서는 침입탐지 시스템의 트래픽 처리에 대한 탐지율 감소를 줄이기 위한 방법으로, 침입탐지 시스템이 사용하는 패킷 캡쳐 모듈 앞 단에서의 소프트웨어적인 필터링 모듈을 제안하였다. 이는 MS Windows 네트워크 드라이버 구조에 기인하여 제안하였는데, 제안된 인터미디엇 드라이버는 침입탐지 시스템의 패킷 캡쳐 모듈보다 더 하위 계층에 위치하게 되어, 침입탐지 시스템에 우선하여 패킷을 필터링 할 수 있다. 이를 위해 우선 인터미디엇 드라이버를 개발하였고, 개발된 인터미디엇 드라이버에 Hacking-Free-Packet을 필터링 할 수 있는 필터 모듈을 추가하였다. 필터 모듈은 ASL을 이용하여 CFG 모델로 구현함으로서 처리 속도 및 안정성을 높였다.

인터미디엇 드라이버에서 Hacking-Free-Packet 필터링에 대한 침입탐지 시스템이 기존의 침입탐지 시스템보다 탐지 성능향상을 측정하기 위해, 인터넷 트래픽 중 가장 많은 비중을 차지하는 HTTP 트래픽에서의 Hacking-Free-Packet을 정의하고, 이를 필터링 할 수 있도록 필터 모듈을 제안하였다. 우선, 제안된 필터 모듈은 100Mbps의 트래픽을 처리할 수 잇는 패킷 처리속도와 안정적인 동작을 보여주었다. 침입 탐지 성능을 측정하기 위해, 100Mbps 네트워크 환경에서 전체 대역폭의 10% 에서 99%까지의 트래픽을 발생시켰으며, 트래픽중 HTTP 트래픽과 UDP 트래픽의 비율을 변화 시키며 침입 패킷에 대한 탐지율을 측정하였다. 실험 결과에서 보듯이, 제안된 시스템은 네트워크 트래픽이 높아질수록, 기존의 시스템에 비해 탐지율에 대한 성능 향상을 보였으며, 또한, 네트워크 트래픽중 HTTP 패킷이 차지하는 비중이 높아질수록, 제안된 시스템은 기존의 시스템보다 탐지율이 높아짐을 알 수 있었다.

이러한 Hacking-Free-Packet에 대한 필터링은 침입탐지 시스템의 탐지 성능을 향

상시킬 수 있는 반면, 제안된 Hacking-Free-Packet을 이용한 침입은 탐지 할 수 없는 단점을 가지고 있다. 그러므로, Hacking-Free-Packet에 대한 계속적인 분석 및 갱신이 필요하고, 이를 유연성 있게 필터링 모듈에 추가 할 수 있는 시스템에 대한 연구 및 디자인이 필요할 것이다.

참고 문헌

ACME Labs, http_load (multiprocessing http test client)

Anonymous, 2000, 리눅스 보안의 모든 것, 인포북 SAMS

Apache, http://www.apache.org

CERTCC-KR, http:/www.certcc.or.kr

HantechSnS, http://hantech.cheju.ac.kr

Charles Iheagwara and Andrew Blyth, “Evaluation of the performance of ID systems in a switched and distributed environment: the RealSecure case study”, 2002

Guang Yang, A real tim packet filtering module for network intrusion detection system, 1998

Insecure, http://www.insecure.org/sploits/land.ip.DOS.html

Jongwook Moon 2001, Enhancing IDS performance through dropping hacking-free packets, Ajou University

Karanjit S. Siyan, 1998, TCP/IP 완전정복, 성안당

K. Thompson, G.J. Miller, and R. Wilder, “Wide-Area Internet Traffic Patterns and Characteristics, *1997*

MSDN, http://msdn.microsoft.com

Mier Communication Inc, “Lab Testing Summary Report”, 2001

Naval Research Laboratory, “MGEN –3.2 User’s Guide”

PCAP, http://www.tcpdump.org,

PLUS, 2000, Seucrity PLUS for UNIX, 영진.com

RobertGraham, http://www.robertgraham.com/pubs/network-intrusion-detection.html

Rebecca Gurley Bace, 2000, Intrusion Detection, MACMILLAN TECHNICAL PUBLISHING

SANS Institute, http://www.sans.org/newlook/resources/IDFAQ/ID_FAQ.htm

S. McCanne and V.Jacobson. “The BSD Packet Filter: A New Architecture for User-level Packet Capture”, 1993

Snort, http://www.snort.org

Winpcap, http://winpcap.polito.it/

Willam Stallings, 1997, 통신망 정보보호, 도서출판 그린

W. Richard Stevens, 1999, TCP/IP 네트워크, 도서출판 진영사

김상철, 2001b, abnormal IP Packets, CERTCC-KR

김영균, 2001, 계층적인 에이전트 기반의 침입탐지 구조 설계

방성민, 2000, 이동에이전트 기술을 이용한 네트워크 패킷 기반 침입탐지 시스템

이현우, 2000b, 네트워크 공격기법의 패러다임 변화와 대응방안 – Part I : 네트워크 공격기법의 패러다임 변화 v1.0-, CERTCC-KR

이현우, 2001, 네트워크 공격기법의 패러다임 변화와 대응방안 – Part II : 대응방안 v0.1-, http://www.securitymap.net/

감사의 글

길었던 2년간의 대학원 생활을 이렇게 마무리 짓고 있습니다. 어찌 보면 짧았다고 할 수도 있는 석사과정 동안, 많은 이야기 거리와 추억이 있었습니다. 다소 후회도 남지만 앞으로의 인생에 좋은 경험으로 남을 기간 이었습니다. 이에 많은 조언을 아끼시지 않았던 분들에게 이렇게 감사의 글을 올립니다.

우선, 따끔한 충고와 격려를 아끼시지 않으셨던 김경식 교수님께 깊은 감사를 드립니다. 또한 이 논문을 제작할 수 있도록 많은 도움을 주신, 고성택 교수님, 김경연 교수님에게도 감사 드립니다. 그리고, 저의 대학원 생활을 지켜봐 주시며 많은 질타와 충고를 아낌없이 주신 이광만 교수님, 도양회 교수님, 강민제 교수님에게도 깊은 감사 드립니다.

대학원 생활동안 친구, 혹은 선배로서 많은 도움을 주신, 경희 누나, 유신 누나, 은성이 형, 봉석이 형, 영균이 형, 숙인, 혜진, 은미에게도 감사드리며 기쁨을 함께 나누고 싶습니다.

또한, 이 논문을 쓰는데 아낌없는 조언을 해준 한텍 식구들인 승학이 형, 영기 형, 철규 형, 은지 누나, 두혁, 보승이에게도 정말 고맙다는 말을 전하고 싶습니다.

자주 만나지는 못하지만, 친구로서 많은 고민을 털어놓을 수 있도록 술친구가 되 준, 민석, 정훈, 영은, 동필, 군석, 경용, 성혁, 철우, 승화, 성호, 에게도 고마움을 전합니다. 특히 논문 마무리 기간동안 결혼한 동헌, 수진부부에게도 앞으로 행복하게 살길 바란다는 말을 전해 주고 싶습니다. 그리고, 저를 위해 항상 기도해 주는 미경에게도 감사를 드립니다.

늘 곁에 있지만 소중함을 몰라 소홀했던, 형과 동생 선옥에게도 이 고마움을 전합니다. 그 밖에도, 여기에 적지 못했지만 많은 도움을 주신 주위 모든 분들에게도 감사를 드립니다.

마지막으로, 둘째 아들을 희생과 사랑으로 보살펴 주신 부모님, 세상에서 가장 존경하고 사랑하는 부모님께 이 논문을 바칩니다.

2002. 12.

김성윤