고속 망에 적합한 네트워크 프로세서 기반 인-라인 모드 침입탐지 시스템

(A Network Processor-based In-Line Mode Intrusion Detection System for High-Speed Networks)

강 구 홍 † 김 익 균 †† 장 종 수 ††
(Koohong Kang) (Ikkyun Kim) (Jongsu Jang)

요 약 본 논문은 ASIC에 상응하는 성능을 가지며 일반 프로세서에 상응하는 유연성을 지닌 네트워크 프로세서(NP: Network Processor)를 사용하여 인-라인 모드 네트워크 기반 침입탐지시스템(NIDS: Network-based Intrusion Detection System)을 제안한다. NP를 이용한 다양한 네트워크 응용들이 제안되고 있으나, NIDS에 직접 적용한 예는 아직 없다. 제안된 NIDS는 패킷 차단과 트래픽 미터링 뿐만 아니라 공격을 검출하기 위해 패킷 내용을 검색한다. 특히, 2-레벨 탐색 기법은 패킷 차단과 트래픽 미터링 기능을 복잡하고 많은 시간을 요하는 패킷 내용 검색 기능과 분리시킴으로서 인-라인 모드 시스템의 성능, 안전성, 그리고 확장성을 향상시켰다. 한편 PC 플랫폼과 Agere PayloadPlus (APP) 2.5G NP를 사용한 프로토-타입을 구현하였고, APP NP에 적용될 패킷 내용 검색 알고리즘을 제안하였다.

키워드 : 침입탐지시스템, 네트워크 프로세서, 공격 시그너처

Abstract In this paper, we propose an in-line mode NIDS using network processors(NPs) that achieve performance comparable to ASIC and flexibility comparable to general-purpose processors. Even if many networking applications using NPs have been proposed, we cannot find any NP applications to NIDS in the literature. The proposed NIDS supports packet payload inspection detecting attacks, as well as packet filtering and traffic metering. In particular, we separate the filtering and metering functions from the complicated and time-consuming operations of the deep packet inspection function using two-level searching scheme, thus we can improve the performance, stability, and scalability of In-line mode system. We also implement a proto-type based on a PC platform and the Agere PayloadPlus (APP) 2.5G NP solution, and present a payload inspection algorithm to apply APP NP.

Key words : Intrusion Detection System, Network Processor, Attack Signature

1. 서 론

오늘날 산업, 정부, 그리고 심지어 일반 개인 생활이 인터넷에 점점 더 의존해 감에 따라 네트워크를 통한 정보 흐름에 관한 중요성이 더욱 강조되고 있다. 그러나 해커들에 의한 네트워크 혹은 주요 서버 컴퓨터에 대한 침입이나 공격은 네트워크를 마비시키거나 서비스거부 공격(DoS: Denial of Service)으로 인해 전자 상거래 서비스 중단과 같은 심각한 경제적 손실뿐만 아니라 인터넷 서비스 중단에 따른 극심한 사회적 혼란을 초래하고 있다. IDS는 이러한 악의적인 공격으로부터 컴퓨팅 시스템 혹은 네트워크를 보호하는 장치이다[1-3].

IDS는 호스트-기반 IDS와 네트워크-기반 IDS (NIDS)로 구별된다. 호스트-기반 IDS는 오디팅 시스템 혹은 이벤트 로그를 이용하여 서버 혹은 개인용 컴퓨터와 같은 하나의 종단 시스템 혹은 네트워크 애플리케이션을 보호한다. 반면, NIDS는 네트워크 트래픽을 모니터링하여 공격 혹은 침입을 감지해 네트워크 관리자에게 보고하고, 한 단계 더 나아가 이를 차단하는 기능까지 수행한다(일명, 침입방지시스템 IPS: Intrusion Prevention System 이라고 한다). 특히 네트워크 기반 공

† 정 회 원 : 서원대학교 컴퓨터정보통신공학부 교수
khkang@seowon.ac.kr
†† 비 회 원 : 한국전자통신연구원 보안게이트웨이연구팀 연구원
kimik@etri.re.kr
jsjang@etri.re.kr
논문접수 : 2003년 12월 30일
심사완료 : 2004년 3월 30일

격이 지난 수년간 꾸준히 늘고 있어 네트워크 보호 구조에 있어 NIDS가 대단히 중요한 위치를 차지하고 있다.

NIDS는 다음 세 가지 분야, 즉 공격 시그너처 검출, 어노멀리 검출, DoS 검출 중 하나의 분야에 집중해 개발되고 있다[3]. 해커들은 이전에 성공적으로 사용된 공격 방법을 이용해 네트워크를 공격한다. 이러한 공격들은 네트워크 보안 제품 생산자들에 의해 분석되어지고 자세한 프로화일 혹은 공격 시그너처가 만들어진다. 시그너처 검출 기술들은 네트워크 트래픽 내에 공격 지문을 조사하고 알려진 시그너처와 비교함으로서 네트워크 공격 혹은 침입을 검출해낸다. 일단 공격 시그너처가 입력 트래픽 내에 확인되면 NIDS는 알람 혹은 경보 신호를 발생하여 네트워크 관리자가 이를 인지할 수 있도록 한다. 그러나 오늘날 급격하게 증가하는 네트워크 대역폭으로 인해 스노트[8]와 같은 순수 소프트웨어 기반 NIDS 사용은 불가능해 졌다. 즉 범용 프로세서의 기술 발전이나 메모리와 같은 서브 시스템 연결 기술들은 기가비트 인터넷 인터페이스 속도를 따라가지 못하고 있는 실정이다[4-6].

최근 고속 NIDS를 위해 트래픽 분산[7] 과 ASIC을 이용한 새로운 방법들이 제안되고 있다[8]. 그러나 전자의 경우 동적 부하 균형을 위한 장치와 여러 개의 센서로 인해 시스템이 방대해지고, 후자의 경우 ASIC 개발 기간으로 인해 표준과 요구사항 변화에 신속하게 대응하지 못한다. 본 논문은 ASIC에 상응하는 성능을 가지며 일반 프로세서에 상응하는 유연성을 지닌 NP를 사용하여 인-라인 모드 NIDS를 제안하였다. 제안된 NIDS는 보안의 첫 단계인 방화벽[9] 기능을 위한 패킷 차단, 다양한 트래픽 통계자료 및 DoS 검출[12-14]을 위한 근거 자료 제공을 위한 트래픽 미터링, 그리고 공격 시그너처 검출이 가능한 NIDS에 초점을 맞춘다.

본 논문은 서론에 이어 제2장에서는 고속 NIDS에 관련된 연구동향과 문제점을 간략히 살펴보고, 제3장에서는 2-레벨 계층적 구조의 인-라인 모드 NIDS를 제안하고 장점을 논하였다. 제4장에서는 APP NP를 이용한 시그너처 매칭을 위한 기본 알고리즘을 제시하고, 제5장에서는 프로토-타입 구현을 위한 방법을 자세히 설명하였다. 제6장에서는 프로토-타입을 통해 얻어진 실험 결과와 시뮬레이션 결과를 논하고, 마지막으로 제7장에서 결론을 맺는다.

2. 고속 NIDS 관련 연구동향

서론에서 언급한 바와 같이 네트워크 대역폭의 급격한 증가로 인해 고속 NIDS에 대한 관심이 집중되고 있다. 고속 NIDS 구현을 위한 현 추세는 크게 두 가지, 즉 트래픽 분산에 의한 여러 센서들의 부하 분담 방법[7]과 ASIC 형태의 하드웨어 가속기를 사용한 성능 향상 방법[8]이 있다. 참고문헌 [7]은 고속 입력 트래픽 스트림을 여러 개의 작은 스트림으로 나누어 분산된 센서로 전송하고 이들 각 센서들은 침입탐지 시나리오 중 일부만을 책임짐으로서 실시간으로 동작하게 된다. 그러나 트래픽 분산을 위한 스케터(scatterer), 슬라이서(slicer), 그리고 스위치가 필요하며, 여러 개의 동일 센서가 분산됨에 따라 NIDS 전체 시스템 구성이 방대해진다. 뿐만 아니라, 트래픽 스트림의 대역폭과 검색할 시그너처 수가 NIDS 성능을 현저하게 변화시킴으로 NIDS 시스템 구성에 어려움이 있다.

참고문헌 [2]는 알테라 EP 20K 시리즈 FPGA를 이용해 2.88 Gbps 네트워크 스트림을 검색하는 장치를 제시하였다. 현재 논의되고 있는 대부분의 고속 NIDS는 이와 같은 하드웨어 가속기를 사용한 성능향상 방법이다. 그러나 이들 ASIC을 이용한 하드웨어 가속기는 관련 표준안이나 시스템 요구 사항 변경 시 새로운 ASIC 개발 기간으로 인해 시스템에 적용까지 어려움을 겪을 수밖에 없다. 더욱이, 네트워크 침입과 공격이 날로 다양하고 지능화 되고 있어 이에 대한 신속한 대처가 무엇보다 중요한 문제로 대두되고 있다. 따라서 단순히 ASIC에만 의존하는 방법 또한 한계가 있다.

궁극적으로 현 시점에 요구되는 NIDS는 Gbps 이상의 트래픽을 선로속도로 처리하고 시그너처 수가 시스템 성능에 영향을 미치지 않아야 한다. 또한 관리자가 실시간으로 시그너처를 추가 혹은 삭제할 수 있어야 하며, 나아가 특정 트래픽을 차단할 수 있는 기능을 지닌 고속 NIDS를 본 논문을 통해 제안하고자 한다.

3. 인-라인 모드 NIDS를 위한 2-레벨 계층 구조

참고문헌 [3]은 NIDS를 네트워크 상에 어떤 형태로 설치할 것인지에 관한 다양한 토폴로지 - 스위치드 포트 분석, 허브 모니터링, 탭 모드, 인-라인 모드, 그리고 포트 클러스링 - 를 제시하였다. 인-라인 모드 NIDS는 그림 1과 같이 두 네트워크 데이타 패스 사이에 위치하여 실시간으로 악의적인 트래픽을 검출하고 차단함으로서 네트워크를 공격으로부터 보호할 수 있다. 오늘날 대부분 보안장비들은 하나의 시스템 내에서 공격 검출과 차단을 통합하는 추세에 있어 인-라인 모드 NIDS가 선호되고 있으며 본 논문에서도 인-라인 모드를 채택하였다.

인-라인 모드 NIDS는 다음과 같은 두 가지 시스템 요구사항을 가진다. 먼저, 인-라인 모드 NIDS는 두 네트워크 패스 상에 위치함에 따라 이들 네트워크 사이 정상적인 트래픽 흐름을 방해하지 않아야 한다. 즉 하나의 네트워크로부터 입력된 트래픽은 다른 한쪽 네트워크로 선로속도를 유지한 채 출력되어야 한다. 두 번째,

그림 1 인-라인 모드 NIDS

인-라인 모드 NIDS는 그림에서와 같이 단일 장애 점으로서 고장 혹은 성능저하가 발생되면 네트워크 서비스에 심각한 영향을 미치게 된다. 이를 극복하기 위해 인-라인 모드 NIDS는 안정적이고 신뢰성 있는 구조를 채택하여야 한다.

상기와 같은 시스템 요구사항을 충족시키기 위해 본 논문에서는 그림 2와 같이 2-레벨 계층구조를 제안한다. 입력포트를 통해 들어온 트래픽은 레벨-1을 경유해 출력포트로 포워딩된다. 레벨-1은 데이타링크, 네트워크, 그리고 트랜스포트 계층 PDU 헤드 정보를 기준으로 패킷 차단과 트래픽 미터링을 수행한다. 추후 설명되어질 NP의 패턴 매칭 능력은 상기와 같이 PDU의 고정 위치의 패턴 매칭에 효과적이기 때문에 레벨-1의 모든 동작은 선로속도로 이루어진다.

그림 2의 레벨-2는 공격 시그너처를 검출하기 위해 전체 패킷 내용을 검색한다. 앞에서 설명한 바와 같이 이들 공격 시그너처들은 보안업체들에 의해 정리되고 있다. 특히 스노트는 패킷 내 임의의 위치에 존재하는 시그너처를 검출해내는 libpcap을 사용하는 대표적인 제품이다[10,11]. 스노트는 이를 위해 룰 리스트를 사용하며, 본 논문에서 제안된 NIDS 역시 스노트 룰 리스트를 사용했음을 밝힌다.

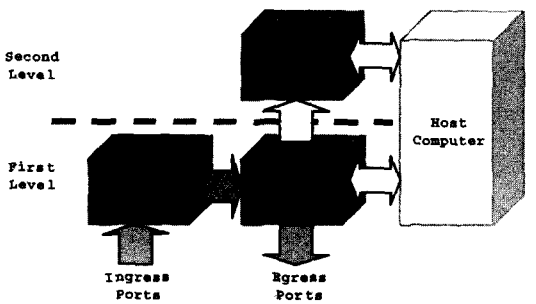


그림 2 인-라인 모드 NIDS의 2-레벨 계층적 구조

그림 3은 스노트가 사용하는 룰의 한 예를 보여준다. 그림에서와 같이 하나의 룰은 두개의 논리적 섹션, 즉 룰 헤드와 룰 옵션으로 이루어진다. 룰 헤드는 룰 액션, 프로토콜, 송신지와 목적지 IP 주소와 네트워크 마스크, 그리고 송신지와 목적지 포트 정보로 이루어진다. 한편 룰 옵션은 경보 메시지와 패이로드 검색을 위한 내용을 포함한다. 따라서 그림 3의 예를 통해, 프로토콜은 TCP, 송신지 IP 주소와 송신지 포트 주소는 모든 값(임의의 값), 목적지 IP는 192.168.1.0 네트워크, 목적지 포트 주소는 111, 그리고 패킷 패이로드 내 2진 데이타 0x000186a5가 존재하는 경우 "mountd access" alert 신호를 발생하게 된다.

alert tcp any any -> 192.168.1.0/24 111
(content:"|00 01 86 a5|" ; Msg: "mountd access";)

그림 3 스노트 룰(밑줄 친 영역: 룰 헤드, 중괄호 영역: 룰 옵션)

그림 2에서 네트워크를 통해 입력되는 PDU들은 패킷 내용을 검색하는 레벨-2로 모두 전달될 필요는 없다. 즉 그림 3에서 룰 헤드에 해당되는 PDU 만 레벨-2로 전송되면 된다. 이와 같이 레벨-1은 룰 규칙에 해당되는 PDU 만을 선택적으로 레벨-2로 전달하게 되며 본 논문에서는 이를 "패킷센싱" 기능이라고 칭한다. 패킷센싱 기능 역시 PDU의 헤드 정보만을 참조함으로 NP가 이들 기능을 선로속도로 처리하는 것에는 전혀 문제가 없다. 그러나 레벨-2에서 수행되는 시그너처 검색은 PDU 전체를 검색해야 하기 때문에 레벨-2가 선로속도로 트래픽을 처리할 수 없는 성능상의 문제가 발생된다. 본 논문에서는 레벨-2에 두 개의 논리적 패킷검색 엔진을 구성함으로서 성능저하 문제를 해결하였다. 즉 하나의 PDU에 대한 패킷 검색은 두개로 병렬화되어 동시에 진행된다. 이러한 접근은 앞에서 설명한 동적부하균등에 의한 트래픽 분산 기법과 유사한 방법이라고 할 수 있으나, 본 논문에서는 NP를 사용함에 따라 시스템 구성이 방대해지는 것을 막을 수 있다. 한편, 제안된 계층적 구조는 레벨-1과 2를 상호 독립적으로 개발할 수 있어 시스템 개발과 확장성에도 뛰어난 효과가 있다.

4. APP NP를 이용한 시그너처 매칭을 위한 기본 알고리즘

4.1 APP NP의 패턴 매칭

현재 세계적으로 40개 이상의 기업에서 다양한 NP를 제작하고 있으며 이에 따른 네트워크 응용들이 제안하고 있다[4-6, 17-19]. 이들 중 인텔의 IXP 계열 NP[4]와 어기어의 APP 계열 NP[17-19, 23, 24], 그리고 IBM의 PowerNP 계열 NP[6]에 대한 연구와 응용들이 많이 발표되고 있다. 그러나 아직 이들 NP들이 NIDS에 직접 적용된 예를 찾아볼 수는 없다. 그 이유는 이들 NP가 PDU(Protocol Data Unit) 내 특정 위치에 고정된 데이타 필드를 검색하는 응용에는 유리한 반면,

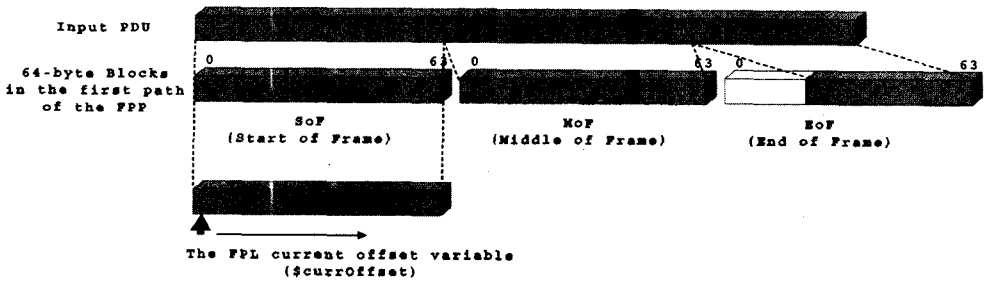


그림 4 FPP의 데이타 프로세싱

PDU 내의 임의의 위치에 존재하는 시그너처를 검출하는 응용에는 적합하지 않기 때문이다. 즉 PDU 헤드 내 목적지 주소와 같은 고정위치 정보에 의한 고속 라우터 개발은 매우 적합한 반면, PDU 내 특정 시그너처의 존재 여부를 확인하기 위한 전체 패킷 검색은 NP 구조에 따라 심각한 성능 저하가 초래된다. 따라서 NIDS 응용에 적합한 NP를 먼저 선정하고, 해당 NP의 기능을 효과적으로 활용할 수 있는 NIDS 구조가 제시되어야 한다. NIDS 응용을 위한 NP 선정문제는 여러 고려사항들이 있어 쉽게 결론 내릴 수는 없다. 그러나 앞에서 언급한 패킷 차단, 트래픽 미터링, 그리고 시그너처 매칭을 위해 PDU 내 패턴검색이 가장 중요하므로 NP의 패턴 매칭 능력이 NP 선정에 가장 중요한 요소로 고려되어야 한다. 본 논문에서는 패턴 매칭에 사용되는 패턴 수가 NP 성능에 영향을 미치지 않고, FPL(functional programming language)[20]로 작성된 제어프로그램에 의해 편리하게 패턴 매칭 기능을 제어할 수 있는 APP NP를 사용한다.

APP NP 2.5G 솔루션은 세 개의 네트워크 프로세서, 즉 FPP(Fast Pattern Processor)[17], RSP(Routing Switching Processor)[18], 그리고 ASI(Agere System Interface)[19]로 이루어진다. 이 중에서 패턴 매칭은 FPP에서 이루어진다. 이때 FPP는 두개의 패스로 PDU들을 처리하는데, 그림 4에서와 같이 패스 1은 입력 PDU를 64 바이트 블록 단위로 데이타를 프로세싱하고, 패스 2는 패스 1이 해당 PDU의 마지막 블록(EoF: end of frame)을 처리하면 전체 PDU 단위로 데이타를 프로세싱한다.

FPL 프로그램에서 함수 기술은 기본 프로세싱 툴이다. 즉 FPL 프로그램의 패턴 매칭과 데이타 스트림 처리는 함수로서 정의된다. 여기서, 함수는 다음 그림 5와 같이 함수명과 함수 컴포넌트라고 불리는 하나 이상의 함수 혹은 상수로 이루어진다. 그림 5에서, 함수 기술은 왼쪽에서부터 오른쪽으로 순서대로 5개의 컴포넌트를 수행하고 만약 어떤 컴포넌트를 성공적으로 마치지 못

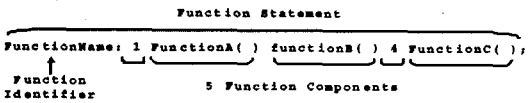


그림 5 함수 구문 구조

하면 멈추게 된다. FPL 프로그램은 *current offset* 변수($currOffset)를 사용하여 블록 내에서 읽어 들일 다음 바이트 위치를 추적한다(그림 4 참조). 따라서 FPL 프로그램이 데이타를 처리할 때 $currOffset은 자동으로 증가하게 된다.

4.2 시그너처 검색을 위한 패턴 매칭의 문제점

먼저 IP 페이로드에 12바이트 길이의 공격 시그너처 0x123456789ABCDEF012345678가 PDU 내에 존재하는지 검색한다고 가정한다. 4.1절에서 설명한 바와 같이 FPP는 패스 1과 2에서 각각 독립적으로 패턴 매칭이 이루어진다. 먼저, 패스 2를 이용해 검색할 경우 그림 6과 같은 FPL 함수 구문이 사용된다.

```
GoPayload: fSkip(160) SignatureDection();
SignatureDetection: 0x123456789ABCEF012345678 fReturn(1);
SignatureDetection: BITS:96 fRSkip(88) SignatureDetection();
```

그림 6 패킷 처리 FPL 예

그림 6에서 fSkip()과 fReturn()는 FPL 내부 함수이며, fSkip(160) 함수는 $currOffset을 데이타 스트림의 시작에서부터 160 비트를 이동시켜 IP 페이로드 시작점으로 이동시키는 역할을 한다. 이후 IP 페이로드의 시작 바이트를 기준으로 12 바이트 길이의 공격 시그너처 검색을 위한 매칭과정이 수행되면, $currOffset는 데이타 스트림의 시작으로부터 256 비트 위치로 이동된다. 그러나 만약 해당 시그너처가 발견되지 않으면 $currOffset을 IP 페이로드의 시작점을 기준으로 두 번째 바이트로 이동시켜 다시 검색해야 한다. 이때 fRSkip(88) FPL 내부 함수를 사용하여 $currOffset을 데이타 스트림의 역으로 이동시킬 수 있다. 그러나 fRSkip(x) 함수는 패

스 2에서 64바이트를 경계로 이 전 블록으로 넘어갈 수가 없다. 따라서 해당 PDU의 길이가 65 바이트보다 크면, 패스 2에서 루프를 돌면서 반복적으로 해당 시그너처의 존재 여부를 검색하는 것이 불가능해 진다.

한편, 패스 1에서 시그너처를 검색할 경우, 그림 4에서와 같이 64 바이트 블록으로 나뉘어 처리되기 때문에 공격 시그너처가 PDU 내 블록 경계(블록과 블록 사이)에 존재할 경우 사실상 검색이 불가능해 진다.

4.3 시그너처 매칭을 위한 기본 알고리즘

PDU 내 임의의 위치에 존재할 공격 시그너처를 검색하기 위해 FPP 패스 2를 이용하는 것은 FPP의 구조상 불가능하다. 따라서 본 논문에서는 4.2절에서 언급한 FPP 패스 1의 문제점을 극복함으로서 시그너처 매칭을 구현한다. 또한 공격 시그너처 길이는 32 바이트 이하로 가정하여 기본 알고리즘을 최대한 간략화하였다.

먼저 그림 7과 같이 입력 PDU(이하, BPDU라 칭함)의 처음 32 바이트가 제거된 PDU(이하, SPDU라 칭함)를 하나 생성하고, 64 바이트 각 블록을 기준으로 앞쪽 32바이트는 BPDU에서, 그리고 뒤쪽 32 바이트는 SPDU에서 시그너처 존재 여부를 검색한다. 그러나 블록 경계에 존재하는 시그너처를 검색할 수 있게 되는 반면, APP NP가 처리해야하는 트래픽 양이 두 배 증가하는 단점이 발생된다. 이러한 단점은 APP 10G NP[24]로 극복될 수 있음을 시뮬레이션 결과를 통해 검증하였다(제6장 참조).

5. 구 현

5.1 NIDS 구조

그림 8은 APP 2.5G NP를 이용해 제3장에서 제안된 2-레벨 계층구조 NIDS를 구현한 블록도다. LIM(Line Interface Module)과 NPM-1(Network Processor Modeule-1)은 그림 2의 레벨-1에 관련된 기능을, 그리고 NPM-2는 레벨-2에 관련된 기능을 각각 구현하였다. 이들 세 종류의 모듈은 PCI 버스를 통해 호스트 컴퓨터와 연결되고 호스트 컴퓨터는 NIDS 운영을 위한 다양한 애플리케이션을 구동한다.

5.2 패킷 필터링과 트래픽 미터링

대부분 사람들이 네트워크 보안을 이야기할 때 방화벽[9]을 생각한다. 방화벽은 보안 구조상 첫 단계 방어 시스템으로 송신지와 목적지 주소를 기준으로 특정 프

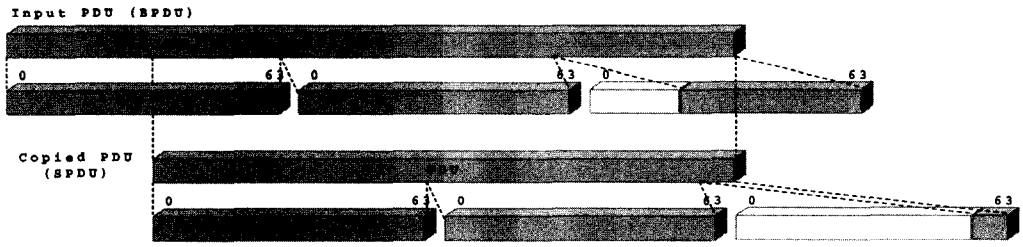


그림 7 BPDU와 SPDU 관계 및 FPP 패스 1의 블록단위 프로세싱

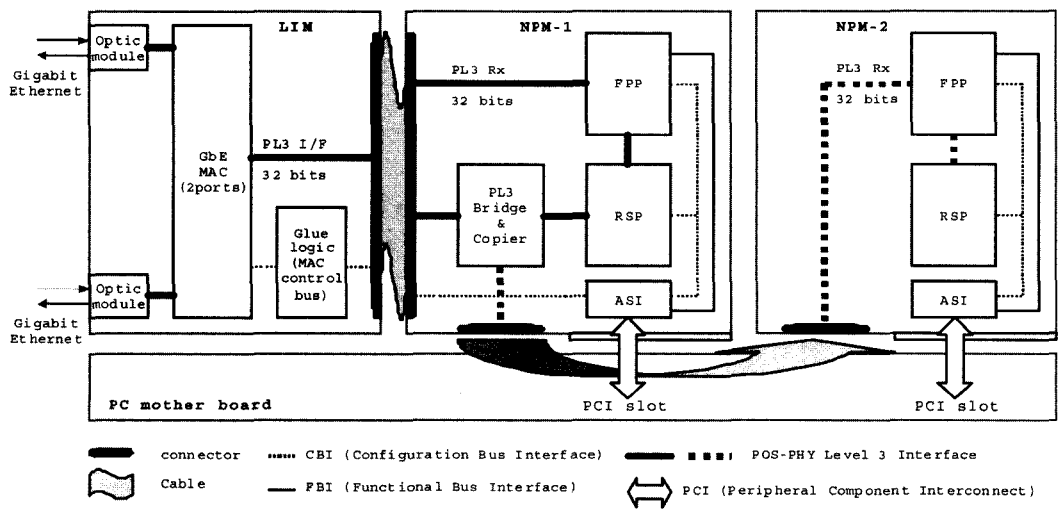


그림 8 APP 2.5G NP 기반 NIDS의 블록도

로토콜들의 접근을 제어한다. 본 논문에서는 공격 검출과 차단의 상호 보완적 관계를 유지하여 보다 안전한 네트워크 보안전략을 수립하기 위해 패킷 필터링을 NIDS 기본기능으로 채택하고 이를 접근제어 리스트(ACL: Access Control List)를 사용해 구현하였다. ACL은 패킷 헤드의 특정 필드로 정의된 규칙의 집합이다. 표 1은 플러딩 DoS 공격 중 하나인 SYN 공격(특정 호스트로 전달되는 TCP SYN 트래픽 흐름)[12-14]을 차단하기 위한 ACL 엔트리를 보여준다. ACL 구현을 위한 입력 트래픽 패턴 매칭은 NPM-1의 FPP 패스 2에서 이루어진다.

트래픽 미터링은 과금[16], 네트워크 설계, 그리고 어노멀리 검출 및 DoS 공격 검출을 위한 각종 자료를 제공한다. ASI는 RMON MIBs(Management Information Bases)와 같은 응용들을 위한 통계수집 용도로 네 가지 함수를 가지고 있다[20]. FPL 프로그램 내 카운터 함수는 트래픽 통계를 위한 카운터를 제공하며 ASI에 의해 제공되는 FBI(Function Bus Interface) 함수 중 하나다. 그림 8의 NPM-1을 구성하는 ASI는 FPP의 FBI를 이용한 함수 호출에 의해 FPP에 의해 처리된 트래픽의 각종 통계 자료들을 실시간으로 관리한다. 표 2와 3은 인터페이스와 흐름에 관련된 통계 수집에 대한 간단한 예를 각각 보여 준다.

패킷 차단 및 트래픽 미터링을 위한 표 1, 2, 그리고 3의 각 엔트리들은 호스트 프로세서에서 운영 중인 애플리케이션 프로그램에 의해 실시간으로 추가 혹은 삭제되며, ASI에 의해 수집된 각종 통계 자료들은 PCI 버스를 통해 호스트 프로세서로 전달된다[23]. 또한 상기 두 기능은 PDU 헤드 내 특정 위치에 존재하는 데이타에 대한 검색 작업으로 FPP의 패턴 매칭 능력에 아주 적합하며 이들 엔트리는 수백만 개까지 지원 가능하다[17].

5.3 시그너처 검색

제4장에서 설명한 바와 같이 PDU 내 공격 시그너처를 검색하기 위해, NPM-1은 패킷센싱 기능과 BPDU/SPDU 생성을 위한 멀티캐스팅을 수행하고, NPM-2는 실질적인 스그너처 검색을 수행한다. 제 3장에서 언급한 바와 같이, 본 논문에서는 스노트 룰을 사용하며 그림 9에 예시한 네 가지 룰을 사용해 구체적인 구현 방법을 설명한다.

그림 9에 나타난 스노트 룰을 구현하기 위해, NPM-1은 표 4의 왼쪽 칸에 보여지는 PDU 헤드 내 고정위치에 존재하는 정보를 기준으로 패킷 센싱 여부를 결정한다. 한편, NPM-1의 RSP는 센싱된 PDU에 대해 그림 10과 같이 룰 아이디(Rule ID)를 추가하고, PL3& Copier는 BPDU와 SPDU를 생성된다. 룰 아이디는 NPM-2에서 해당 시그너처를 매칭을 위한 FPL 함수 구문을 구현하기 위해 사용된다. 그림 9의 (예3)은 단지 헤드 내 정보를 이용해 NPM-1에서 공격 검출이 가능하기 때문에 패킷이 센싱될 필요가 없다. 따라서 룰 아이디가 필요치 않다. 한편, (예1)과 (예2)는 패킷의 페이로드 내 시그너처는 다르지만 헤드 내 정보는 동일함으로 동일 룰 아이디를 갖게 된다.

앞에서 언급한 바와 같이 NPM-2는 FPP 패스 1에서 시그너처 검출을 통해 네트워크 침입을 검출하게 된다. 따라서 그림 4와 같이 BPDU 혹은 SPDU는 패스 1에서 64 바이트 크기의 블록 단위로 나뉘어져 처리되며, 하나의 PDU는 시작블록(SoF: Start of Frame), 마지막 블록(LoF: Last of Frame), 그리고 중간블록(MoF: Middle of Frame)로 구성된다. 그림 11은 NPM-2의 FPP 패스 1에서 PDU 내 시그너처를 검색하기 위한 흐름도이다. 각 블록들은 싱글 블록(S&EoF: Start and End of Frame, 하나의 PDU가 하나의 블록으로 구성된 경우), SoF, MoF, 그리고 EoF에 따라 패턴 매칭이 이루어진다.

그림 11에서 시작블록인 경우 글로벌레지스터에 룰

표 1 ACL 리스트 예

protocol	source address	destination address	source port	destination port	flag	action
TCP	*.*.*.*	192.208.12.20	*	21	SYN	deny
*	*.*.*.*	*.*.*.*	*	*	*	allow

표 2 인터페이스 통계 수집 예

타입	이름	설명
인터페이스 통계	IfInOctects	수신된 옥텟의 전체 갯수
	IfInUcastOkts	클래스 A, B, 그리고 C 목적지 IP 주소를 가진 수신 패킷의 전체 수

표 3 흐름 통계 수집 예

프로토콜	발신지 주소	목적지 주소	발신지 포트	목적지 포트	플래그
TCP	*.*.*.*	192.208.12.20	*	21	SYN
TCP	210.110.100.*	192.208.12.*	*	*	*

```
#---------
# X11 RULES, X11 MIT cookie (예1)
#---------
alert tcp $EXTERNAL_NET any -> $HOME_NET 6000 (msg:"X11 MITcookie"; flags: A+ content:
"MIT-MAGIC-COOKIE-1"; reference:arachnids,396; classtype:bad-unknown; sid:1225; rev:1;)
#---------
# X11 RULES, X11 xopen (example: RuleIDGroup) (예2)
#---------
alert tcp $EXTERNAL_NET any -> $HOME_NET 6000 (msg:"X11 xopen"; flags: A+; content: "|6c00 0b00
0000 0000 0000 0000|"; reference:arachnids,395; classtype:unknown; sid:1226; rev:1;)
#---------
# X11 RULES, X11 outgoing (example: RuleIDSimple) (예3)
#---------
alert tcp $EXTERNAL_NET 6000:6005 -> $HOME_NET any (msg:"X11 outgoing"; flags: SA;
reference:arachnids,126; classtype:unknown; sid:1227; rev:1;)
#-------------------------------------
# Subseven22 is a Trojan Horse (example: RuleIDNew) (예4)
#-------------------------------------
alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any(msg:"BACKDOOR subseven 22";
flow:to_server,established; content:"|0d0a5b52504c5d3030320d0a|"; reference:arachnids,485;
reference:url,www.hackfix.org/subseven/; classtype:misc-activity; sid:103; rev:5;)
```

그림 9 스노트 룰 예

표 4 스노트 룰 (그림 9참조) 기준으로 만들어진 룰 아이디

프로토콜	발신지 주소	목적지 주소	발신지 포트	목적지 포트	플래그	룰 아이디	시그너처
TCP	external_NET	home_NET	*	6000	A	0x0001	X_11MITcookie X11_open
TCP	external_NET	home_NET	6000:6005	*	SA	N/A	X11_outgoing
TCP	external_NAT	home_NET	27374	*	*	0x0002	Backdoor

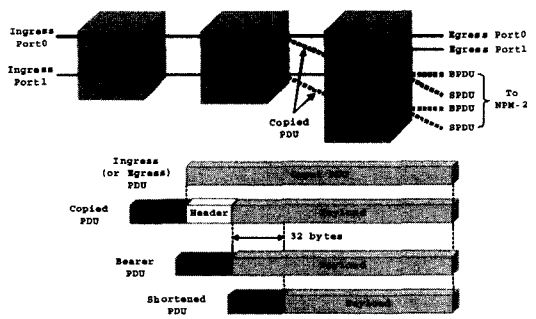


그림 10 NPM-1의 패킷 센싱 및 BPDU/SPDU 생성

아이디를 저장한다. 이것은 중간블록이나 마지막블록이 룰 아이디를 포함하고 있지 않기 때문에 FPL 프로그램이 이들 블록을 처리할 때 해당 시그너처 트리 함수를 찾기 위해 글로벌레지스터에 저장되어 있는 룰 아이디를 사용하기 위함이다.

그림 12와 그림 13(a)는 센싱된 PDU로부터 싱글블록이 만들어지는 예를 보여주고, 그림 14는 싱글블록 처리 루틴의 상세 흐름도를 나타낸다. 그림 14에서, L_{min}은 동일 룰 아이디에 존재하는 시그너처 중 최소 길이를 표현하고, 따라서 $currOffset+$L_{min}$이 65보다 크거나 같으면 패이로드를 더 이상 검색할 필요가 없어진다. 한편 블록 내 바이트 수를 나타내는 변수 $currLength가 34(SPDU의 경우, 32) 보다 클 경우, 시그너처 검색을 위해 33번(SPDU의 경우 31번) 이상 검색할 필요가 없다. 이것은 BPDU(혹은 SPDU)의 나머지 부분은 SPDU(혹은 BPDU)에서 검색되기 때문이다(그림 12(b)와 그림 13(a) 참조).

그림 15는 센싱된 PDU로부터 FPP 패스 1에서 시작블록, 중간블록, 그리고 마지막블록이 만들어지는 과정을 보여준다. 그림 16은 시작 혹은 중간블록 처리 흐름을 나타낸다. 이들 블록들은 64 바이트 고정 길이로 구성된다. 따라서 BPDU 시작 블록을 제외하고(그림 13(b) 참조) $currOffset가 32가 될 때까지 패이로드를 검색하면 된다. 그림 17은 마지막 블록 처리 흐름을 보여준다.

6. 실 험

6.1 프로토-타입 구조

본 연구에서는 제안된 NIDS의 성능을 평가하기 위해

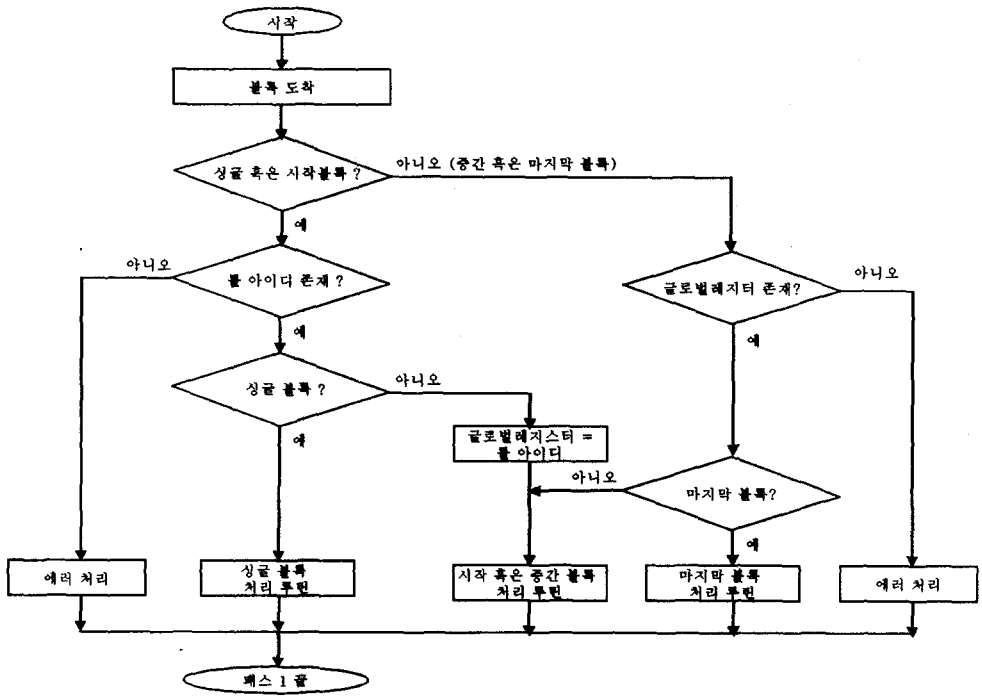
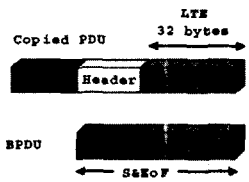
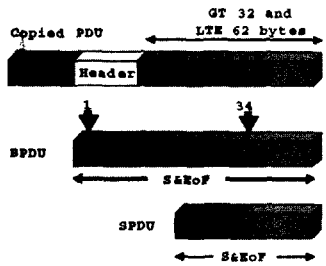


그림 11 NPM-2 FPP 패스 1에서 시그너처 매칭 흐름도

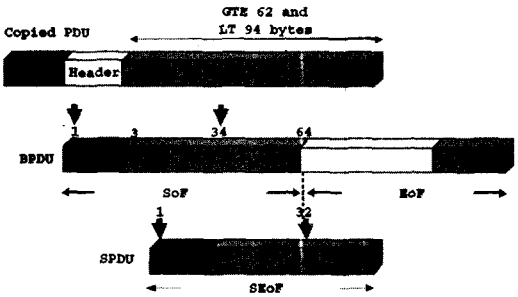


(a) 싱글블록으로 구성된 BPDU 예

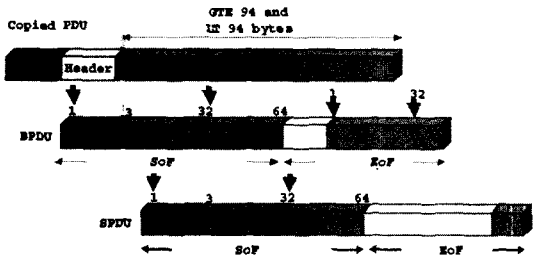


(b) 싱글블록으로 구성된 BPDU와 SPDU 예

그림 12 싱글블록 생성 예



(a) 시작블록과 마지막블록으로 구성된 BPDU, 싱글블록으로 구성된 SPDU 예



(b) 시작블록과 마지막블록으로 구성된 BPDU와 SPDU 예

그림 13 시작블록, 마지막블록, 그리고 싱글블록 생성 예

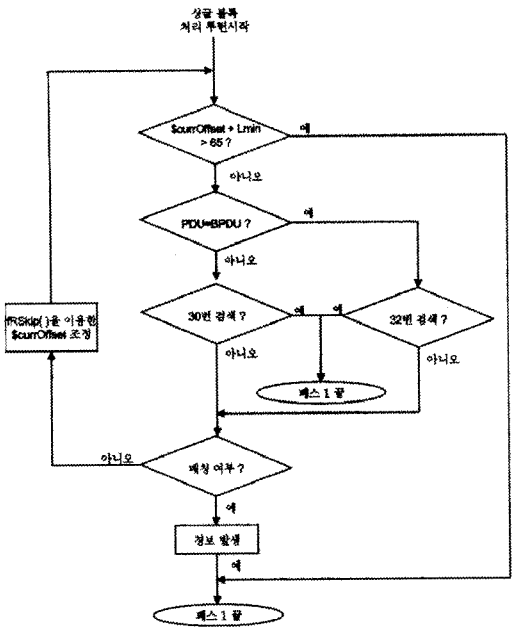


그림 14 싱글블록 처리 상세 흐름도

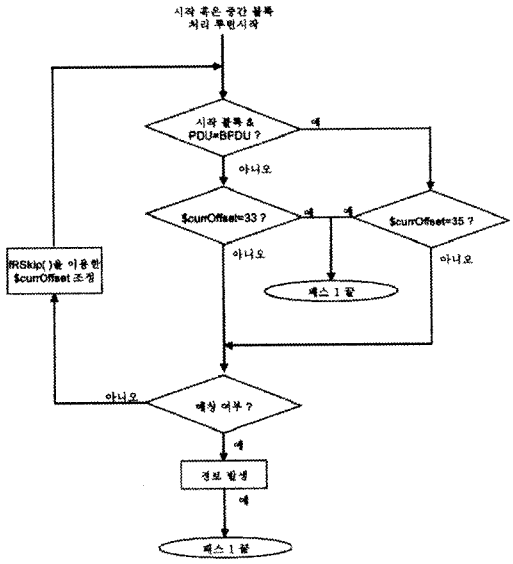


그림 16 시작 혹은 중간블록 처리 흐름도

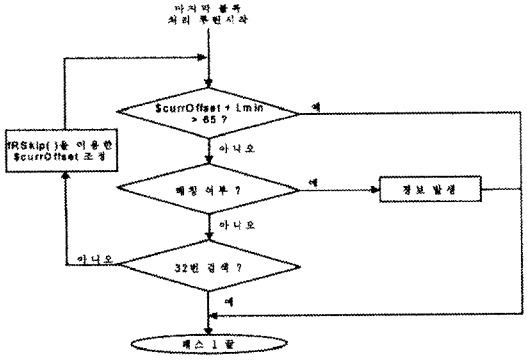
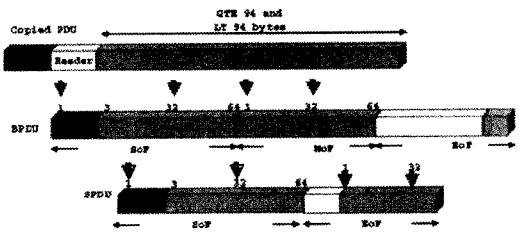


그림 17 마지막 블록 처리 흐름도

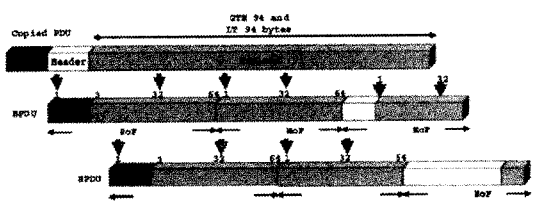
리눅스 운영체제를 사용하는 산업용 PC 플랫폼과 APP 2.5G NP를 사용한 두 장의 회로기판(라인카드와 NP 카드)을 제작하여 프로토-타입 시스템을 구현하였다(그림 18 참조). 라인 카드는 두 개의 외부 기가비트 이더넷 포트를 수용하고, ASI에 의해 지원되는 PCI 접속을 통해 PC 상의 호스트 프로세서와 상호 연결된다.

NP 카드는 FPP와 RSP를 이용해 만들어지고 패킷 필터링, 트래픽 미터링, 그리고 시그너처 검색 기능을 수행한다. NP 카드는 커넥터를 사용하여 라인카드 위에 탑재되고 호스트 프로세서와의 정보전달은 라인카드를 통해 이루어진다. 그림 18에서 보여지는 바와 같이 라인 카드와 NP카드 쌍으로 이루어진 NPM-1과 NPM-2가 산업용 PC의 PCI 슬롯에 삽입된다. 한편, NPM-1과 NPM-2 상의 두 NP 카드는 mictor 커넥터와 임피던스 매칭을 지원하는 5 인치 길이의 블루리본 케이블을 이용해 센싱된 PDU를 104Mbps 병렬 전송한다.

그림 18에서 라인카드와 Shomiti의 THGs(Ten Hund-

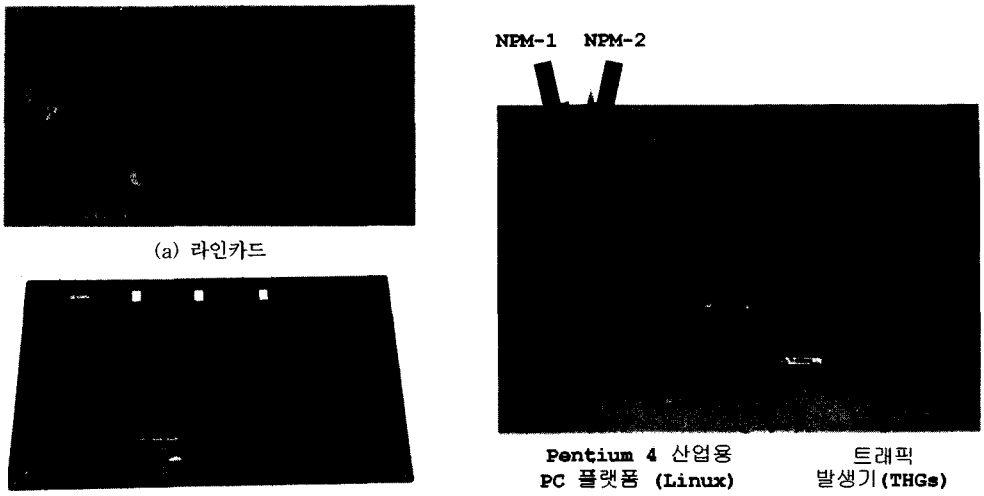


(a) 시작블록, 중간블록, 마지막블록으로 구성된 BPDU, 시작블록과 마지막블록으로 구성된 SPDU 예



(b) 시작블록, 중간블록, 마지막블록으로 구성된 BPDU와 SPDU 예

그림 15 시작블록, 중간블록, 그리고 마지막블록 생성 예



(a) 라인카드

(b) NP 카드

(c) 프로토-타입 시스템

그림 18 실험 환경

red Gigabit system)[25] 사이 연결된 두개의 기가비트 링크는 양방향 1Gbps 이더넷 트래픽을 제공한다. 이 경우, THGs는 선로속도로 이더넷 트래픽을 생성하지만 실질 트래픽은 이더넷 프레임 사이 12 바이트 간격 시간으로 인해 1Gbps 선로속도 보다 작게 된다. 한편, THGs는 그림 9에 예시한 네 가지 공격 트래픽을 발생시킨다.

6.2 실험 결과

실험은 먼저 패킷 필터링과 트래픽 미터링의 기능과 성능을 검증하였다. Shomiti의 THGs에 64 바이트 IP 패킷을 1Gbps 선로속도로 전송하는 두개의 TCP 연결을 설정하고, 이들 트래픽을 각각 두 개의 기가비트 이더넷 포트로 동시에 전송하였다. 앞에서 언급한 바와 같이 64 바이트 IP 패킷의 실질 트래픽은 이더넷 프레임 사이 시간으로 인해 선로속도의 76%에 해당된다. 제작된 프로토-타입은 상기 실험 환경에서 선로속도로 패킷 필터링과 트래픽 미터링 기능을 완벽하게 처리하였다.

두 번째 실험은 시그너처 검색에 관련된 기능과 성능을 검증하였다. Shomiti의 THGs는 그림 9에 예시된 (예1), (예2), 그리고 (예4) 공격 트래픽을 각각 생성하였다. 이때 패킷 길이에 따른 성능변화를 관찰하기위해 64, 128, 256, 512, 그리고 1024 바이트 길이의 패킷에 대해 각각 성능을 관찰하였다. 또한 공격 시그너처는 발생 PDU 페이로드의 끝에 위치해, NPM-2가 시그너처 검색에 소요되는 시간을 최대로 하였다. 따라서 다음에 제시될 분석결과들은 최악의 시나리오에서 얻어진 결과들이다. 그러나 이러한 공격 트래픽은 1Gbps 선로속도로 하나의 이더넷 포트로만 전송된다. 이것은 NPM-1에서 이들 공격 트래픽이 모두 센싱될 경우 NPM-2로 2Gbps 센싱 트래픽이 전달되고 이러한 트래픽 양은 APP 2.5G NP로 제작된 프로토-타입이 처리할 수 있는 최대 양이다. 한편 NPM-2의 시그너처 검색에 관한 성능을 관찰하기 위해 호스트 프로세서에서 운영되고 있는 침입차단모듈(IPM: intrusion prevention module)을 비 활성화하여 공격 트래픽이 NPM-1에서 차단되지 않고 계속 NPM-2로 센싱되도록 하였다(IPM은 NPM-2로부터 경고 메시지를 수신하면 NPM-1의 ACL에 해당 엔트리를 추가하여 공격을 차단하는 IDS 운영 애플리케이션 중 하나).

표 5는 NPM-2가 공격 시그너처를 검색하는 최대 쓰루풋을 보여준다. 표 5에서 실질적인 1Gbps 이더넷 트

표 5 NPM-2의 패킷 길이에 따른 시그너처 검색 최대 처리율 (1Gbps 선로속도 기준)

패킷길이 (바이트)	1Gbps 트래픽 (%)	2.5G APP 실험치 (%)	2.5G APP 시뮬레이션 (%)	5G APP 시뮬레이션 (%)
64	76	32	19	46
128	86	34	31	55
256	92.7	34	37	62
512	96.2	34	41	68
1024	98	35	40	67

래픽은 프레임 사이 간격으로 인해 패킷 길이가 길어지면 따라서 증가한다. 한편 프로토-타입을 통해 관찰된 APP 2.5G NP의 경우 패킷 길이에 상관없이 34%의 쓰루풋을 보였다. SPA[21]를 이용한 APP 2.5G NP 시뮬레이션 결과와 측정된 결과가 표 5에서와 같이 다소 차이가 있으나 SPA를 이용한 APP 5G NP[23] 시뮬레이션 결과로부터 60% 정도의 쓰루풋을 예상할 수 있다. 이러한 성능 향상 추세로 보아, APP 10G NP[24]를 이용해 NIDS를 설계할 경우 선로속도로 기가비트 이더넷 트래픽 내 공격 시그너처를 검색할 수 있을 것으로 예상된다.

7. 결 론

본 논문은 ASIC에 상응하는 성능을 가지며 일반 프로세서에 상응하는 유연성을 지닌 NP를 사용하여 기가비트 대역폭을 처리하는 NIDS를 제안하였다. 제안된 NIDS는 인-라인 모드 타입으로 시그너처 기반 공격 검출과 패킷 차단을 하나의 시스템에 구현하였으며, 각종 트래픽 통계 수집을 위한 트래픽 미터링 기능을 구현하였다. 또한 제안된 2-레벨 탐색 기법은 인-라인 모드 시스템의 성능, 안전성, 그리고 확장성을 향상시켰으며, NP를 이용함에 따라 네트워크 관리자는 실시간으로 패킷 필터링, 트래픽 미터링, 그리고 시그너처 관련 룰들을 갱신할 수 있다. 특히 본 논문에서는 APP NP를 이용해 PDU 내 공격 시그너처를 검색하기 위한 알고리즘을 제시하였으며 APP 2.5G NP를 사용해 프로토-타입 시스템을 직접 제작해 성능을 검증하였다.

프로토-타입 시스템을 이용한 시험 결과는 첫 번째 레벨에서 두 개의 기가비트 포트의 전체 트래픽을 안정적으로 차단 혹은 계량함을 보였으며 두 번째 레벨에서 320Mbps까지 패킷 내용을 실시간으로 검색하였다. 그러나 10Gbps APP NP를 사용할 경우, 시뮬레이션 결과는 선로속도로 2 Gbps까지 패킷 내용 검색이 가능함이 예상된다. 한편, 시그너처 검색 수가 증가할 경우 FPL 함수 구문 수가 증가하게 되나, 이러한 현상은 APP NP의 특성상 성능에는 전혀 영향을 미치지 않는다.

참 고 문 헌

[1] McHugh, J. Christie, A. and Allen, J., "Defending Yourself: The Role of Intrusion Detection Systems," IEEE Software Magazine, Sept./Oct. 2000.

[2] Debar, H. Dacier, M. and Wespi, A., "Towards a taxonomy of intrusion-detection systems," Computer Networks, Vol.31, No.8, pp. 805-822, 1990.

[3] Gong, F., "Next Generation Intrusion Detection System (IDS)," IntruVert Networks Report, 2002.

[4] Memik, G. and Maggion-Smith, W.H., "NEPAL: A Framework for Efficiently Structuring Applications for Network Processor," Proc. Second Workshop on Network Processors, 2003.

[5] Memik, G. Mangion-Smith, W.H. and Hu, W. "NetBench: A Benchmarking Suite for Network Processor," Proc. ICCAD, pp. 39-42, 2001.

[6] Allen, J.R., "IBM PowerNP network processor: Hardware, software, and applications," IBM J. RES. & DEV. vol. 47, No. 2/3, 2003.

[7] Kruegel, C. Valeur, F. Vigna, G. and Kemmerer, R., "Stateful Intrusion Detection for High-Speed Networks," Proc. IEEE Symposium on Security and Privacy, 2002.

[8] Cho, Y.H. Navab, S. and Maggione-Smith, W.H., "Specialized Hardware for Deep Network Packet Filtering," Proc. Field Programmable Logic and Applications (FPL), 2002.

[9] Ranum, M.J., "Thinking about Firewalls," Proc. SANS-II, 1994.

[10] Roesch, M., "Snort Lightweight Intrusion Detection for Networks," Proc. USENIX LISA'99, pp. 101-109, 1999.

[11] Roesch, M., Snort Users Manual Snort Release:1.8, Snort, 2001.

[12] Ferguson, P. and Senie, D., "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," RFC 2267, http://www.ietf.org, 1998.

[13] Wang, H. Zhang, D. and Shin, K.G., "Detecting SYN Flooding Attacks," Proc IEEE INFOCOM 2002, 2002.

[14] Moore, D. Voelker, G. and Savage, S., "Inferring Internet Denial of Service Activity," Proc. USENIX Security Symposium'2001, 2001.

[15] Stevens, W.R., TCP/IP Illustrated, Volume 1, Addision-Wesley, Massachusetts 1994.

[16] Hartanto F. and Carle, G., "Policy-Based Billing Architecture for Internet Differentiated Services," Proc. IFIP Fifth Int. Conference on Broadband Communications, 1999.

[17] Agere Systems Inc., PayloadPlus Fast Pattern Processor, http://www.agere.com, 2001.

[18] Agere Systems Inc., PayloadPlus Routing Switch Processor, http://www.agere.com, 2001.

[19] Agere System Inc., PayloadPlus Agere System Interface, http://www.agere.com, 2001.

[20] Agere Systems Inc., Functional Programming Language User's and Reference Guides for SDE Version 3.0, 2001.

[21] Agere Systems Inc., PayloadPlus Simulator User's Guide, 2001.

[22] Agere System Inc., PayloadPlus Application Programming Interface User's Guide, 2001.

[23] Aagere System Inc., Technical Guide to the APP550 and APP530 Network Processors, 2003.

[24] Agere System Inc., Technical Guide to the APP750TM Preliminary Data Book, 2003.
[25] A Finisar Company, Shomiti THGs; Distributed 10/100/1Gb Network QoS System, http://www.shomiti.net/shomiti/thgs.html.

부 록

본 부록은 그림 9에 예시한 시그너처를 기준으로 작성된 FPL 소스 코드를 보여준다.

```
SETUP ROOT                  (MapPort);

////////////////////////////////////////////////////////////////////////////////
//                          Pass 1 : MapPort
////////////////////////////////////////////////////////////////////////////////

MapPort:    asIncrement32(ETHERNET_PKTS_SECOND)
            EtherBlock($framerEOF:1,$framerSOF:1);

EtherBlock:  RANGE(0b10, 0b11):2 PayloadInsp() ProcessLastBlock();// Start and End of Frame, or EOF
EtherBlock:  BITS:2 PayloadInspMiddle() ProcessBlock();     // Start of Frame or Middle Frame

ProcessBlock: fQueue(0:2, $PortNumber:16 , $offset:6, 0:1, 0:1, 0:1);
ProcessLastBlock: fQueueEOF(0:2, $PortNumber:16 , $offset:6, 0:1, 0:1, 0:1, P0_DIRECTION);

////////////////////////////////////////////////////////////////////////////////
//                          Searching the RuleID based on the Snort's Rule header
////////////////////////////////////////////////////////////////////////////////

 PayloadInsp:       0x0001 X11($curLength);
 PayloadInsp:       0x0002 BackDoorsubseven22($curLength);
// At this line, add a component of tree PayloadInsp for a RuleIDView

 PayloadInspMiddle: 0x0001 X11();
 PayloadInspMiddle: 0x0002 BackDoorsubseven22Middle();
// At this line, add a component of tree PayloadInspMiddle for a RuleIDView

////////////////////////////////////////////////////////////////////////////////
//                          Searching the Start and/or Last Blocks
//
//      Inspect until $curOffset = 64 - Lmp, where Lmp is the length of matching pattern.
//      For example, Lmp = 12 in case of X11Open. So inspect until $curOffset = 0x34:6 (=52)
//                   Lmp = 18 in case of X11MITcookie. So inspect until $curOffset = 0x2e(=46)
////////////////////////////////////////////////////////////////////////////////
// X11
////////////////////////////////////////////////////////////////////////////////

X11:    RANGE(0x00000, 0x0008):18 fReturn();
X11:    RANGE(0x0000A, 0x0011):18 X11IIIOne();
X11:    BITS:18 X11III();

X11III:  0x6c000b0000000000000000 BITS:48 asIncrement32( X11RULE_xopen) fReturn();
X11III:  0x4d49542d4d414749432d434f 0x4f4b49452d31 asIncrement32(X11RULE_MITcookie) fReturn();
X11III:  BITS:96 BITS:48 fRSkip(136) fSub(0x2e:6, $curOffset:6)
CheckStillLoop($4:6) fBmz($4:1,X11III) X11IIIOne();

X11IIIOne:      0x6c000b0000000000000000 asIncrement32( X11RULE_xopen) fReturn();
X11IIIOne:      BITS:96 fRSkip(88) fSub(0x34:6, $curOffset:6)
CheckStillLoop($3:6) fBmz($4:1,X11IIIOne) fReturn();

////////////////////////////////////////////////////////////////////////////////
// BackDoorsubseven22
////////////////////////////////////////////////////////////////////////////////

BackDoorsubseven22:     RANGE(0x00000, 0x0008):18 fReturn();
BackDoorsubseven22:     BITS:18 BackDoorsubseven22III;

BackDoorsubseven22III:  0x0d0e5b52504c5d3030320d0a
asIncrement32(BackDoorsubseven22) fReturn();
BackDoorsubseven22III:  BITS:96 fRSkip(88) fSub(0x34:6, $curOffset:6)
CheckStillLoop($3:6) fBmz($4:1, BackDoorsubseven22III) fReturn();

////////////////////////////////////////////////////////////////////////////////
// At this line, add a tree for a RuleIDView
////////////////////////////////////////////////////////////////////////////////

CheckStillLoop:  0x00:6 fReturn(0b0:1);
CheckStillLoop:  BITS:6  fReturn(0b1:1);
```

```
////////////////////////////////////////////////////////////////////////////////
//                          Searching the SOF or Middle Blocks
//
//                  Inspect until $curOffset = 31, where 31 means the center of a 64-byte block
////////////////////////////////////////////////////////////////////////////////

X11Middle:      0x6c000b0000000000000000 BITS:48 asIncrement32( X11RULE_xopen) fReturn();
X11Middle:      0x4d49542d4d414749432d434f 0x4f4b49452d31
asIncrement32(X11RULE_MITcookie) fReturn();
X11Middle:      BITS:96 BITS:48 fRSkip(136) fSub(0b100000:6, $curOffset:6)
CheckStillLoop($4:6) fBmz($5:1,X11OpenMiddle) fReturn();

BackDoorsubseven22Middle: 0x0d0e5b52504c5d3030320d0a
asIncrement32(BackDoorsubseven22) fReturn();
BackDoorsubseven22Middle: BITS:96 fRSkip(88) fSub(0b100000:6, $curOffset:6)
CheckStillLoop($3:6) fBmz($4:1, BackDoorsubseven22Middle) fReturn();

// At this line, add a tree for a RuleIDView
```

강 구 홍

1985년 경북대학교 전자공학과 졸업(공학사). 1985년~1993년 한국전자통신연구소 선임연구원. 1990년 충남대학교 전자공학과 졸업(공학석사). 1998년 포항공과대학교 전자계산학과 졸업(공학박사). 1998년~1999년 한국전자통신연구원 선임연구원. 2000년~2001년 서원대학교 정보통신공학과 전임강사. 2002년~2003년 한국전자통신연구원 초빙연구원. 2002년~현재 서원대학교 컴퓨터정보통신공학부 정보통신공학과 조교수. 관심분야는 성능평가, 컴퓨터 네트워크, 네트워크 보안

김 익 균

1994년 경북대학교 컴퓨터공학과 졸업(공학사). 1996년 경북대학교 컴퓨터공학과 졸업(공학석사). 1996년~1998년 한국전자통신연구소 연구원. 1999년~2000년 (주) 팍스콤 선임연구원. 2001년~현재 한국전자통신연구소 선임연구원. 관심분야는 컴퓨터 네트워크, 네트워크 보안

장 종 수

1984년 경북대학교 전자공학과 졸업(공학사). 1986년 경북대학교 전자공학과 졸업(공학석사). 2000년 충북대학교 대학원(공학박사). 1989년 7월~현재 한국전자통신연구원 네트워크보안그룹장. 관심분야는 네트워크 보안, 센서네트워크, 정책기반보안관리, QoS