시그너처 해싱에 기반한 고성능 침입방지 시스템

왕정석[O] 권희웅 정윤재◆ 곽후근 정규식
숭실대학교 정보통신 전자공학부
{wang[O], hukwon, gobarian, kchung}@q.ssu.ac.kr, jgyver@gmail.com◆

A High Performance IPS Based on Signature Hashing

Jeongseok Wang[O] Huiung Kwon Yunjae Jung Hukeun Kwak Kyusik Chung
School of Electronics Engineering, Soongsil University

요 약

침입방지 시스템(IPS, Intrusion Prevention System)은 인라인모드(in-line mode)로 네트워크에 설치되어, 네트워크를 지나는 패킷 또는 세션을 검사하여 만일 그 패킷에서 공격이 감지되면 해당 패킷을 폐기하거나 세션을 종료시킴으로서 외부의 침입으로부터 네트워크를 보호하는 시스템을 의미한다.

침입방지 시스템은 크게 두 가지 종류의 동작을 수행한다. 하나는 이미 알려진 공격으로부터 방어하는 시그너처 기반 필터링(signature based filtering)이고 다른 하나는 알려지지 않은 공격이나 비정상 세션으로부터 방어하는 자기 학습 기반의 변칙 탐지 및 방지(anomaly detection and prevention based on self-learning)이다.

시그너처 기반 필터링에서는 침입방지시스템을 통과하는 패킷의 페이로드와 시그너처라고 불리는 공격 패턴들과 비교하여 같으면 그 패킷을 폐기한다. 시그너처의 개수가 증가함에 따라 하나의 들어온 패킷에 대하여 요구되는 패턴 매칭 시간은 증가하게 되어 패킷지연 없이 동작하는 고성능 침입탐지시스템을 개발하는 것이 어렵게 되었다. 공개 침입방지 소프트웨어인 SNORT를 위한 여러 개의 효율적인 패턴 매칭 방식들이 제안되었는데 시그너처들의 공통된 부분에 대해 한번만 매칭을 수행하거나 한 바이트 단위 비교대신 여러 바이트 비교 동작을 수행함으로써 불필요한 매칭동작을 줄이려고 하였다.

본 논문에서는 패턴 매칭 시간을 시그너처의 개수와 무관하게 하기 위하여 시그너처 해싱 기반에 기반한 고성능 침입방지시스템을 제안한다.

1. 서 론

인터넷을 비롯한 네트워크의 의존도가 높아지고, 점점 더 중요하고 복잡한 작업들이 컴퓨터와 이들의 연결 집합체인 네트워크를 통해 이루어 지고 있다. 이로 인해 네트워크 트래픽은 급증하여 이미 수 기가비트 급으로 확장되었고, 이러한 네트워크 속도의 증가로 인해 네트워크를 통한 서비스 거부 공격(DoS, DDoS), 인터넷 웜(internet worm), 이메일 바이러스, 피싱(phishing)등의 악의적인 공격도 빠른 속도로 증가하고 있으며, 이로 인한 피해도 급격히 증가되는 추세이다.

네트워크 및 내부 시스템에 대한 악의적인 공격에 대한 방어로 기존에는 방화벽(firewall), 침입탐지 시스템(IDS: Intrusion Detection Systems)등이 사용되었으나, 방화벽이나 침입탐지 시스템은 최근에 많은 문제를 일으키는 웜, 이메일 바이러스 등의 공격 등을 차단하지 못하고, 이미 공격이 진행되어 내부 네트워크가 그 공격으로 인한 피해를 입은 후에 감지 할 수 있어 효과적이고 빠른 대응을 기대하기 어렵다.

침입방지 시스템(NIPS: Network Intrusion Prevention Systems)은 외부 네트워크와 내부 네트워크의 종단에 인라인으로 위치하며, 오가는 모든 패킷을 시그너처(signature)라고 불리는 공격 패턴을 표현한 룰들과 비교 검사하여 해당 패킷의 감염 및 이상 여부를 판단하고, 패킷을 폐기하거나 세션을 종료함으로써 네트워크를 공격으로부터 보호한다.

이를 위해 그 동안 많은 노력을 통해 침입탐지시스템을 위한 효율적인 패턴 매칭 알고리즘이 고안되었고 현재까지 많은 발전이 이루어졌다. 하지만 보다 효율적인 침입방지 시스템 개발을 위해서 다음과 같은 사항이 고려되어야 한다[1].

- 성능의 안정성 : 단위시간당 들어오는 패킷이 증가함에 따라, 또는 적용되는 공격 패턴 룰의 개수가 증가에 따라 시스템의 성능이 크게 저하되지는 않는가?
- 룰 상관관계와 성능 : 룰의 상관관계에 따라 시스템의 성능이 영향을 받지는 않는가?
- 예측 가능한 성능 : 룰의 내용, 개수, 적용시기 등에 따라 항상 안정적이고, 예측 가능한 성능을 보이는가?
- 정확한 탐지 능력 : 적용되는 룰을 이용하여 항상 정확한 탐지를 해 낼 수 있는가?

본 논문에서는 침입탐지시스템을 위한 기존의 패턴 매칭 알고리즘에 대해 소개하고, 해당 알고리즘이 가지고 있는 한계와 문제에 대해 분석하고, 이를 해결할 새로운 패턴 매칭 알고리즘을 제안한다.

제안된 방식에서는 시그너처라고 불리는 공격 패턴 룰의 개수에 무관하게 동작하는 패턴 매칭 성능, 예측 가능한 성능, 기존 알고리즘보다 빠른 처리능력 등을 보임으로써 기존 알고리즘의 한계를 극복하였다.

2. 관련 연구

알려진 공격에 대해 차단하는 시그너처 기반 필터링의 경우 해당 공격에 대한 패턴을 이용하여 패킷 데이터를 조사한다. 이러한 패턴 매칭을 위해서는 공격이 나타나는 패턴에 대해 정확하게 기술된 내용이 존재하여야 하며, 이 전체적인 내용을 룰이라고 한다.

이 룰에는 직접적으로 패킷의 데이터와 비교될 실제 시그너처를 포함해, 시그너처가 적용될 서비스 포트, 룰의 이름, 시그너처가 나타나는 위치, 비교 방식 등에 대한 상세한 정보가 담겨 있다.

일반적으로 룰은 다양한 형태를 통해 기술될 수 있으나 본 논문에서는 침입탐지 시스템에서부터 현재에 이르기까지 많은 사용층을 확보하고 있는 SNORT[2]의 룰 기술 방식을 따르도록 한다.

그림 1은 시그너처와 룰의 상관관계 및 간단한 예제를 나타낸다.

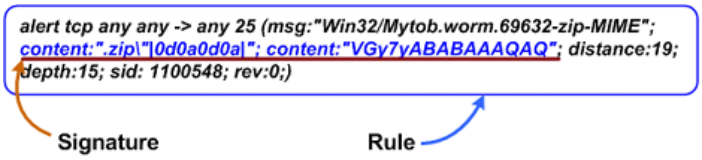


그림 1 시그너처와 룰

2.1 Aho-Corasick 알고리즘

Aho-Corasick 알고리즘[3]은 과거 침입방지 시스템을 위해 고안된 알고리즘으로 기존에 존재하던 텍스트 기반 검색 알고리즘을 혼합하고 수정하여 검색하고자 하는 대상이 여러 개일 때 빠른 검사 및 중복성 검사 등을 해결하기 위해 수정된 알고리즘이다.

비교하려는 시그너처를 등록하는 과정에서 이를 컴파일하여 내부적으로 트리구조를 구성, 트리구조의 매칭을 진행하여 성능을 개선하고자 하였다. 또한 시그너처 간의 상관관계에 주목하여 이들의 상관관계를 이용하여 트리구조 내에서도 시그너처 간에 연관성을 가질 수 있도록 하였고, 매칭이 진행되는 과정에서 트리의 부분 매칭(partial matching) 결과를 저장하여 재사용함으로써 성능 향상을 이룩하였다.

그림 2는 예시된 시그너처들을 검사하고자 할 경우 Aho-Corasick 알고리즘이 이루는 트리 구조 및 매칭 구조를 나타낸다.

그림 2에서도 알 수 있듯이 Aho-Corasick 알고리즘은 트리구조를 통해 매칭 속도와 적용 가능한 룰의 개수를 비약적으로 증진시키는 결과를 보였지만 이는 기본적으로 각 시그너처가 상당한 연관관계를 가져야만 제 효과를 낼 수 있다. 즉 시그너처 간의 상관관계에 굉장히 종속적인 성능을 나타내는 알고리즘으로 불특정 시그너처를 사용하게 되는 실제 네트워크에서 이는 시스템의 성능에 대한 예측을 어렵게 하고, 확장성을 가로막는 요인으로 작용한다.

또한 룰 내의 시그너처간 상관관계가 우수하여 Aho-Corasick 알고리즘이 제 성능을 발휘하는데 적합하다 하더라도, 매칭 알고리즘 자체의 특성상 한 바이트의 캐릭터(character) 당 한 번의 메모리 접근이 필요하며, 한 번에 단 한 바이트만을 매칭 할 수 있다는 점은 결국 메모리 접근과 처리에 병목현상을 불러일으키며, 이로 인한 성능 저하를 피할 수 없게 된다[4].

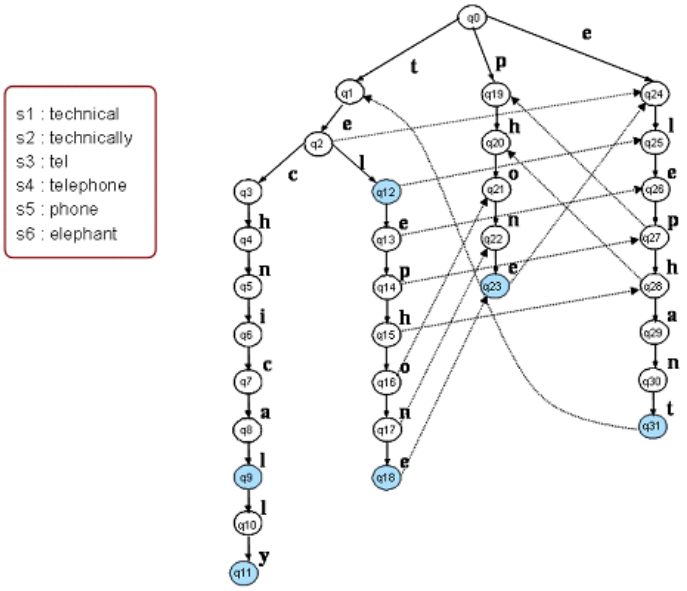


그림 2 Aho-Corasick 알고리즘

2.2 Jump Aho-Corasick 알고리즘

기존의 Aho-Corasick 알고리즘이 트리구조를 통해 성능을 개선한 반면 트리구조를 한 바이트 단위의 각 철자 하나로 구성하고 이에 대한 연관관계를 기술하여 경우의 수가 너무 많다는 점에서 착안 이를 개선한 알고리즘이 Junp Aho-Corasick 알고리즘 이다.

Jump Aho-Corasick 알고리즘은 기존의 Aho-Corasick 알고리즘이 가지는 상태 수를 대폭 축소하고, 시그너처를 트리구조화 하는 과정에서 연관 관계가 있는 철자의 묶음으로 트리를 구성한다.

이를 통해 시그너처와 비교 시 최초 트리 분기점에서 시작한 단어가 블록 단위의 묶음과 일치하지 않을 경우 그 묶음의 길이만큼 건너뛰면서(jump) 비교하여 그 성능이 개선된다는 알고리즘이다.

그림 3에서 보는 바와 같이 Jump Aho-Corasick 알고리즘은 연관성 있는 부분들의 묶음으로 각 시그너처를 조사 및 분해하여 이 분해된 묶음 단위의 블록을 이용하여 트리를 구성한다.

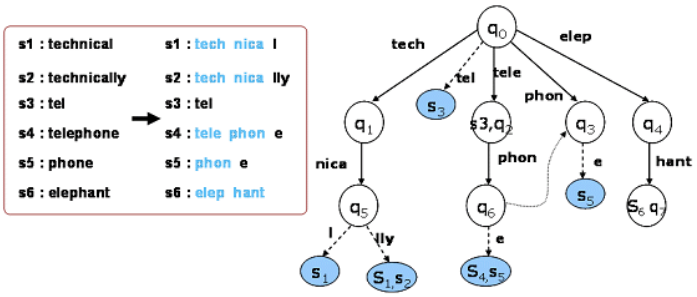


그림 3 Jump Aho-Corasick 알고리즘

Jump Aho-Corasick 알고리즘은 비록 Aho-Corasick 알고리즘이 가지는 메모리 접근의 증가 및 철자 단위의 매칭으로 인한 병목 현상 등의 문제를 어느 정도 완화시키기는 하지만, 실제 존재하는 시그너처의 길이가 매우 다양하고, 그에 따라 많은 메모리를 필요로 하는 단점을 가지고 있다.

그림 3에서 볼 수 있듯이 이는 시그너처의 범위 및 상태 수를 획기적으로 축소하고, 실제 매칭 진행시 건너뛰며 진행될 확률을 높임으로써 성능 개선을 거두었지만, 오히려 이전에 비해 더욱 시그너처 간의 상관관계에 의존적이 되었다.

이는 앞서 지적하였던 Aho-Corasick 알고리즘이 가지는 한계를 그대로 유지하는 것으로 역시 시그너처 간의 관계에 많은 영향을 받으며 성능 수치에 대한 예측이 어려운 알고리즘이다.

3. 시그너처 해싱에 기반한 고성능 침입방지 시스템

패턴 매칭의 수행 성능을 향상시키기 위해서 필요한 가장 중요한 부분은 처리해야 하는 패킷이 도착했을 때, 해당 패킷의 데이터를 룰의 시그너처와 비교하는 횟수를 줄이는데 있다. 즉, 비교해야 하는 룰의 개수를 줄이거나, 비교해야 하는 경우의 수를 줄임으로써 매칭 수행 성능을 향상시킬 수 있다.

하지만 단순히 분류를 잘 해서 매칭 을 수행해야 하는 룰의 수를 줄이는 것은 룰의 개수가 점점 많아지고 있는 현실을 감안하면 그다지 효과적이라고 할 수 없는 방법이다.

이를 보장하기 위해서는 항상 예측 가능하고, 성능이 보장되는 새로운 알고리즘이 필요하고, 또한 앞서 소개한 다른 알고리즘이 가진 한계인 시그너처 간의 상관관계에서도 자유로워야 한다.

3.1 시그너처 해싱

시그너처 해싱이란 적용되는 모든 룰의 내부에 존재하는 시그너처에 대해 이 중 일부(2 bytes)를 정해 해싱 값을 만들고, 이 해싱 값을 이용하여 검색하고자 하는 패킷 데이터와의 비교를 통해 실제 확인해볼 필요가 있는 시그너처 만을 비교 하도록 경우의 수를 현저히 줄이는 방식이다.

시그너처의 특정 2byte를 정하여 이를 해싱의 기준 값으로 삼고, 이 해싱 값에 각 룰의 실제 시그너처를 연결하여 매칭에 사용될 불필요한 동작을 획기적으로 줄임으로써 매칭 속도를 향상시키고, 항상 안정된 성능을 낼 수 있도록 하기 위한 알고리즘이다.

기준치를 2bytes로 삼고 이를 이용하여 해싱 값을 구하는 이유는 일단 해싱을 통해 얻을 수 있는 경우의 수가 충분히 많고, 데이터를 실제 스캔할 때 소요되는 비용이 최소화되기 때문이다.

2bytes를 이용하여 해싱 값을 구할 때 가능한 경우의 수는 이론적으로 모두 65535개 이다. 이는 현재 존재하는 룰의 개수보다 훨씬 많은 양이고, 프로토콜과 포트로 나누어진 상태의 룰에 있어서는 충분히 크기 때문이다.

제안된 방식은 패턴 매칭을 수행하기 전에 적용할 룰을 정하여 해당 룰을 해싱 기법을 이용하여 포트 및 시그너처 해시 테이블로 정리하는 준비 단계와 실제 패킷이 들어왔을 때, 이 패킷 데이터를 조사하여 시그너처가 정확히 존재하는지를 검사하는 부분으로 나누어진다.

3.1.1 포트 / 시그너처 해시 테이블

패턴 매칭 수행에 앞서 적용할 룰에 대해 포트 및 시그너처를 기준으로 정리하는 단계를 거치게 되는데, 이를 위해 룰에서 해당되는 서비스 포트와 시그너처 중 일부(2 bytes)를 추출하여 포트 해시 테이블과 시그너처 해시 테이블을 구성하는 과정이 필요하다.

이를 위해 먼저 각 프로토콜과 서비스 포트 번호 등에 대해 먼저 분류를 하고, 해당 포트를 해싱 한 값을 이용하여 포트 해시 테이블을 구성한다. 포트 해시 테이블을 구성하면, 해당하는 포트 해시 테이블이 갖고 있는 시그너처 해시 테이블에 시그너처의 해싱 값을 등록한다. 여기서 작성한 시그너처 해시 테이블에 실제 매칭에 사용되는 룰의 정보를 연결 리스트를 이용하여 등록하는 과정을 마치면 패턴 매칭을 수행하기 위한 사전 준비 작업을 마치게 된다.

그림 4는 룰에서 포트 해시 테이블과 시그너처 해시 테이블을 구성하는 예와 각 해시 테이블에 어떤 식으로 룰들이 맵핑되는지를 보여준다.

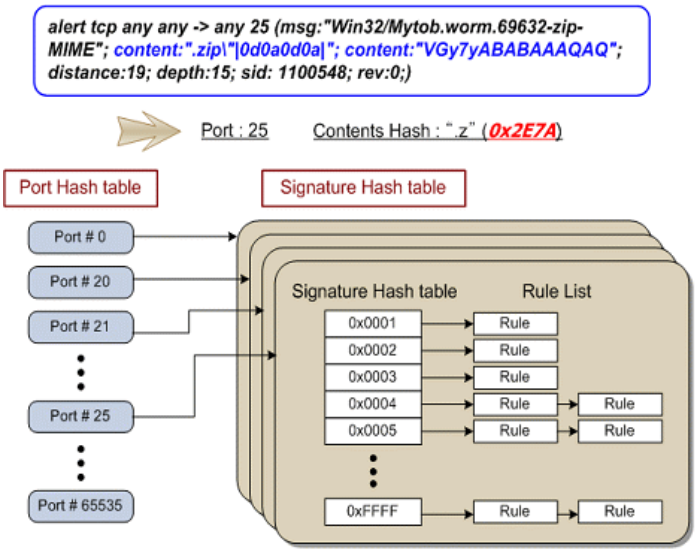


그림 4 포트 및 시그너처 해시 테이블

위와 같이 예시된 룰의 경우 25번 포트(SMTP)를 이용하여 전파되는 이메일 웜 바이러스에 대한 룰로써 일단 포트 정보가 25번이며, 시그너처 해싱 값으로 사용할 부분을 시작 하는 스트링에서 첫 2 bytes를 이용하는 것으로 여기서는 ".z"가 선택되었다는 것을 볼 수 있다. 이 경우 해당 포트별로 룰이 존재할 경우 포트 해시 테이블에 포트의 해싱 값을 등록하고, 해당 포트 해시 테이블에 시그너처 해시 테이블을 연결한다.

시그너처 해시 테이블에 해싱 값을 등록한 후, 그 해싱 값에 룰의 상세 비교 정보를 연결 리스트(linked list)를 이용하여 연결해 놓는다.

이러한 일련의 방법을 통해 적용하고자 하는 모든 룰에 대해 해싱 값을 구하고 연결 리스트를 등록하는 과정을 거치고 나면 포트 해시 테이블과 시그너처 해시 테이블이 완성된다.

해싱 값을 사용하여 검사를 하는 경우 해당 해싱 값이 일치하는지 매칭 하는데 걸리는 소요시간이 일반적인 텍스트 매칭에 비해 극히 짧고, 각 해싱 값에 연결되는 룰의 개수가 줄어듦에 따라 패킷 데이터에 대해 룰과 매칭 되는 경우의 수는 극히 줄어들게 된다.

3.2 패턴 매칭 진행

제안한 방법은 실제 패킷 데이터에 대해 매칭을 수행함에 있어 두 가지 절차를 거치게 된다.

먼저 패킷의 데이터 부분을 기준치(2bytes) 만큼씩 옮겨가며, 해싱 값을 구해 그 해싱 값과 일치하는 값이 시그너처 해시 테이블에 존재하는 지를 확인하는 과정이다. 이 부분은 해싱 기법을 사용하여 매칭을 수행하므로 원하는 해싱 값이 존재하는지를 한 번에 확인할 수 있다.

해싱 값을 얻어 비교하는 데이터 부분을 패킷의 끝부분으로 옮겨가며 계속 검사를 하다가 시그너처 해시 테이블에 존재하는 해싱 값이 발견되면, 이는 해당 패킷이 공격 패킷일 가능성이 있다는 것을 의미하고, 세부 조사를 통해 실제 공격 패킷인지 여부를 정밀하게 검사하여 판단하게 된다.

3.2.1 Phase 1 – Byte windowing

먼저 byte windowing을 통해 패킷의 데이터 부분을 windowing하며 검사한다. 이 경우 시그너처 해싱 값에 사용되었던 기준치(2 bytes) 만큼씩 windowing하며 해싱 값을 구한다. 여기서 구해진 해싱 값을 가지고 이미 작성되어 있는 시그너처 해시 테이블에 같은 값이 있는지를 비교하고, 같은 값이 없다면 해당 부분은 이상이 없다고 판단하고 계속 windowing을 진행한다.

만약 시그너처 해시 테이블과 비교하여 같은 값이 존재하면 해당 두 번째 단계를 적용하여 최종적으로 해당 패킷이 실제 룰과 일치하는지를 정밀 검사한다.

그림 5는 byte windowing을 하며 해싱 값을 찾는 과정을 보여준다.

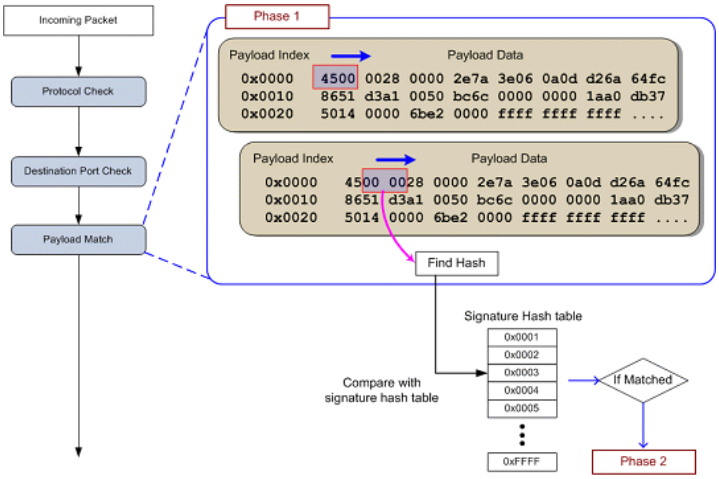


그림 5 Phase 1 – Byte windowing

그림에서와 같이 패킷의 데이터 부분을 byte windowing 하며 2 bytes씩 뽑아내어 해싱 값을 구한 후, 이를 해당 시그너처 해시 테이블과 비교하는 작업을 수행하는 과정을 진행한다. 구한 해싱 값이 시그너처 해시 테이블에 존재하지 않는다면 의심되는 부분이 아니므로 다음 byte로 windowing 하여 다시 해싱 값을 구하고 해시 테이블과 비교하는 작업을 반복한다.

만일 byte windowing을 진행하며 구한 해싱 값이 시그너처 해시 테이블과 일치한 경우가 발생한다면, 이는 공격 패킷으로 의심해 볼 수 있는 상황이 되므로 실제 룰 매칭을 수행하여 세부 검사를 진행한다.

3.2.2 Phase 2 – Rule matching

Byte windowing 하며 얻은 해싱 값과 시그너처 해시 테이블에 존재하는 해싱 값이 같을 경우 해당 패킷이 룰과 일치하는지를 의심하고 확인해 보아야 한다.

이 경우 시그너처 해시 테이블의 해당 해싱 값에 연결된 룰에 대한 연결 리스트를 따라가며 시그너처의 전체 값과 패킷의 해당 데이터를 비교하여 룰에 정확히 일치하는지를 확인한다. 이 작업은 오탐을 막기 위해 굉장히 중요한 행동이며, 룰에 기술된 다른 많은 정보와도 함께 비교하여 정확히 일치하는지를 확인하는 과정을 거쳐 최종적으로 패킷이 룰에 적용되어 적절한 조치를 취해도 되는지를 확인하는 단계이다.

룰의 내부에는 오탐을 막기 위해 출발지와 목적지 정보, 시그너처 이외에도 해당 시그너처가 패킷에서 존재하는 위치 등의 상세 정보가 들어있으며, 이를 통해 더욱 세밀한 검사가 가능하게 한다.

실제 룰 매칭을 수행할 때에는 이러한 부가 정보들의 내용을 먼저 비교하면, 패킷 데이터 부분과 시그너처 전체를 매칭하지 않아도 공격 패킷 여부를 쉽게 파악할 수 있어 더욱 실제 매칭 수행의 가능성을 낮출 수 있다.

그림 6은 룰 매칭을 진행하는 세부 과정에 대해 보여준다.

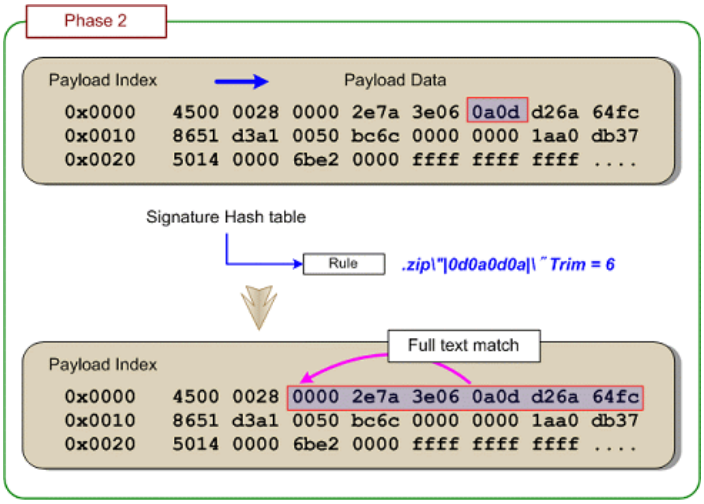


그림 6 Phase 2 – Rule matching

4. 구현 및 실험

4.1 구현 및 실험 환경

제안한 시그너처 해싱 기법을 적용한 알고리즘을 평가하기 위해 실제 침입방지 시스템에서와 같이 리눅스 커널 모듈 형태로 구현하여 실험에 사용하였다.

사용한 커널 버전은 2.6.16.18을 사용하였으며, 리눅스 넷필터(Netfilter)의 PREROUTING hook point를 이용하여 패킷을 처리하는 형태로 구현되었다.

그림 7은 리눅스에서 제안된 알고리즘이 구현되어 적용된 부분에

대해 나타낸다.

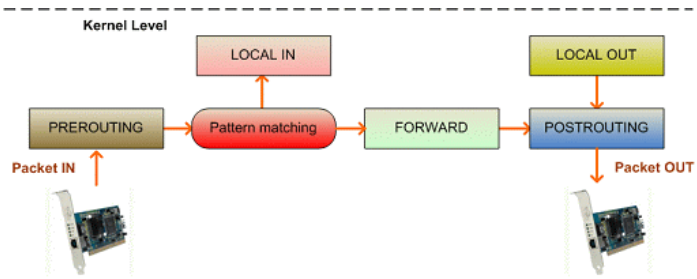


그림 7 패턴 매칭 엔진 구현 위치

표1은 실험에 사용된 하드웨어와 소프트웨어를 나타낸다.

성능과 효율성 측면 뿐만 아니라 네트워크 보안 장비로서 가장 중요한 척도는 역시 공격을 얼마만큼 정확하게 차단하느냐는 부분일 것이다. 따라서 적용되는 룰의 개수에 따른 알고리즘 별 네트워크 트래픽 성능 수치와 알고리즘 별 공격 패킷 검출 성능 수치를 구분하여 실험을 진행하였다[5].

실험을 위해서 웜 패킷을 생산해낼 수 있는 웜 제너레이터와 서버, 그리고 일반적인 네트워크 구성 환경을 시뮬레이션 할 수 있도록 패킷 제너레이터를 이용하여 각각의 환경에 대해 테스트를 수행하였다.

사용자가 웜 제너레이터와 패킷 제너레이터를 이용하여 네트워크 성능에 대한 테스트를 수행하였을 때 기존에까지 연구되어 사용되고 있는 Jump Aho-Corasick 알고리즘을 이용하는 경우와 새로 제안된 시그너처 해싱 기법을 사용한 경우에 대해 룰의 개수에 따른 성능 변화를 측정함으로써 제안된 방법의 성능을 검증하는 방식으로 실험을 수행하였다[6].

표 1 실험용 하드웨어 및 장비

	하드웨어		수 량
	CPU (GHz)	RAM	
웜 제너레이터	P-IV 2.6	512 MB	3
침입방지 시스템	Xeon 3.4	2 GB	1
서버	P-IV 2.6	2 GB	1
패킷 제너레이터	-	-	Smartbit[7] 1EA

기존의 알고리즘과의 비교를 수행함에 있어, 오픈소스 침입탐지 및 방지 시스템으로 널리 사용되고 있는 SNORT와의 비교를 통하여 각 실험 상황에 대해 비교, 평가 하였으며, 사용된 소프트웨어의 항목 및 버전은 표 2와 같다.

표 2 실험용 소프트웨어 항목

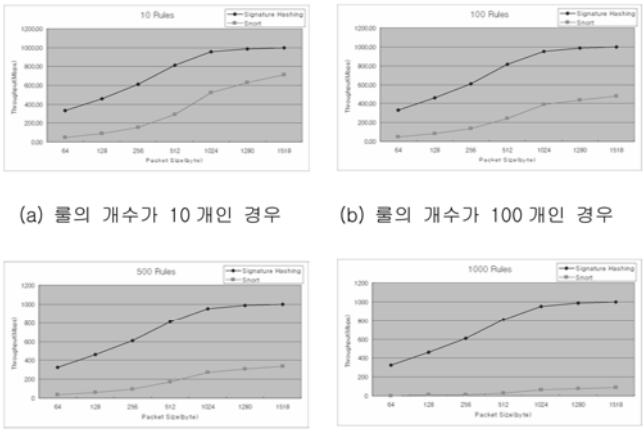
	버 전	수량	비 고
운영체제	2.6.16.18	1	리눅스
SNORT	2.4.5	1	-
룰(Rules)	2006. 8 현재	1000 개	국내 H 사가 제공한 공격 패턴

4.2 패킷 처리 성능 실험

제안된 알고리즘과 기존 알고리즘에 대해 각 패킷 사이즈 별 및 적용 룰 개수 별 패킷 처리 능력에 대한 비교 테스트를 진행하였다. 이를 통해 룰의 확장성과 네트워크 성능 운용에 대한 안정성에 대해 평가 한다.

실험 방법은 적용 룰의 개수를 각각 10, 100, 500, 1000개로 조정해 가며, 각 상황 별로 64~1518 byte의 사이즈를 가진 패킷을 1Gbps의 속도로 전송하여 해당 트래픽에서 처리 여부를 평가한다.

그 결과 그림 8에서 알 수 있듯이 기존의 알고리즘과 제안된 시그너처 해싱 기법을 사용하였을 경우, 룰의 개수가 각각 10, 100, 500, 1000개가 적용되었을 경우에 각 상황별로 패킷을 처리하는 성능을 비교해 보면, 기존 알고리즘의 경우 룰의 적용 개수가 전체적인 패킷 처리 성능에 영향을 미치고 있음을 알 수 있다.



(a) 룰의 개수가 10 개인 경우 (b) 룰의 개수가 100 개인 경우

(c) 룰의 개수가 500 개인 경우 (d) 룰의 개수가 1000 개인 경우

그림 8 룰 개수 및 패킷 사이즈 변화에 대한 처리 성능 비교

제안된 알고리즘의 경우 기존 알고리즘에 비해 룰의 개수가 많아지더라도 전체 네트워크 성능에 큰 차이 없이 고르게 향상된 성능을 보이는 것을 확인할 수 있다.

또한 패킷 사이즈의 변화, 즉 초당 처리 패킷 개수의 변화도 룰의 개수에 관계없이 항상 일정한 형태의 추이를 나타냄으로써 트래픽의 변화에 따른 시스템의 성능에 대해 예측 가능한 모습을 나타낸다.

각 패킷 사이즈 별 차이점은 존재하나 평균적으로 기존 알고리즘에 비해 제안된 알고리즘은 룰의 개수에 대해 10개일 때 약 280%, 100개일 때 약 330%, 500개일 때 약 480%, 2,500%의 성능 향상이 나타남을 확인할 수 있으며, 이를 통해 룰의 개수가 많아짐에 따라 그 성능의 차이는 더욱 더 크게 나타남을 알 수 있다.

4.3 패턴 감지 실험

효과적인 패턴 매칭 알고리즘은 그 처리 속도나 룰 적용 개수와 상관없이 어떠한 경우에도 정확히 패턴에 매칭 되는 패킷을 찾아내어 요구하는 동작을 즉시 수행할 수 있어야 하며, 트래픽 중 일부를 패턴에 매칭 되는 형태로 제작하여 테스트 하였을 경우 정상 패킷의 탐지(false-positive)나 탐지 대상 패킷을 탐지하지 못하는 상황이

발생하는지에 대한 여부가 중요한 평가 기준이 된다.

이를 실험하기 위해 웜 패킷을 생성해내는 웜 제너레이터를 이용하여, 실제 트래픽 중 웜의 비율을 10% ~ 100%까지 변화를 시켜가며 패킷을 보냈을 때, 제안된 알고리즘과 기존 알고리즘이 정확히 패턴을 감지해 낼 수 있는 가를 평가한다.

표 3에 나타난 바와 같이 10%에서 100%에 이르기까지 10%씩 매칭 되는 패턴을 가진 패킷의 비율을 증가시켜가며 패킷을 생성하여 시스템에 보낸 결과 기존의 알고리즘과 제안된 알고리즘 모두 동일하게 패턴을 감지해 냄을 알 수 있다.

이는 제안된 알고리즘이 패턴 감지 능력에 손상 없이 패킷 처리 성능을 향상시켜 신뢰도를 유지하며, 성능을 개선한 알고리즘임을 알 수 있다.

표 3 시그너처 패킷 분포도 및 검출 비율

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Signature Hashing	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Snort	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%

4.4 다양한 형태의 트래픽 처리 실험

제안된 알고리즘과 기존 알고리즘에 대해 적용되는 룰의 개수를 변화시켜 가며, 패킷 제너레이터를 이용하여 다양한 형태의 패킷을 통과시켜 보며 룰의 개수에 따른 네트워크 처리 능력을 실험하였다.

그림 9에서 볼 수 있듯이 기존의 알고리즘은 적용되는 룰의 개수가 증가함에 따라 처리 성능 역시 지속적으로 감소하고 있는 모습을 보인 반면, 제안된 알고리즘은 적용되는 룰의 개수에 관계없이 일정한 성능을 보임으로써 룰 개수에 대한 확장성을 보여준다.

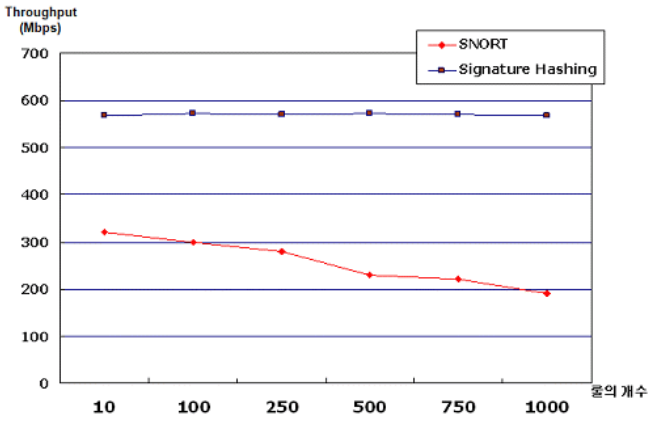


그림 9 다양한 형태의 트래픽에서 패킷 처리 능력 비교

5. 결론 및 향후 연구 방향

본 논문에서는 향후 보다 빠르고 안정적으로 패킷을 검사할 수 있도록 하기 위해 시그너처 해싱 기법을 적용한 알고리즘을 제안하였고, 기존에 사용되던 알고리즘과의 비교를 통해 예측 가능한 높은 성능을 낼 수 있음을 확인하였다.

제안된 시그너처 해싱 기법을 이용하면 룰의 개수에 영향을 받지 않고 실제 시그너처에 수행되는 룰의 경우의 수를 줄임으로써 룰이 적게 걸린 것과 같은 효과를 내서 결국 룰의 개수와 관계없는 예측 가능하고 높은 성능을 내는 것을 확인하였다.

그러나 해싱 기법을 사용하면 룰의 상관관계가 깊을수록 같은 시그너처 해싱 값을 가질 가능성이 높아지고, 하나의 해싱 값에 많은 룰이 연결되면 이는 곧 성능저하와 연결될 가능성이 있다. 또한 기존의 알고리즘에 비해 포트 및 시그너처 해시 테이블을 유지하고 관리함에 따라 기존의 방법에 비해 상대적으로 큰 메모리를 차지하게 되는 측면도 존재한다.

고성능의 침입방지 시스템을 비롯하여 패킷을 깊은 수준에서 분석하고 확인하는데 있어 매칭 알고리즘은 네트워크 성능이 증가하고 그 처리속도가 중요시되는 현실에 비추어 볼 때 매우 중요한 부분임에 분명하다.

이러한 새로운 기법을 이용하면 기존에 비해 훨씬 효율적이고 안정적인 네트워크 보안 구축이 가능하리라 사료된다.

6. 참고 문헌

[1] X. Zhang, C.Li, and W.Zheng, "Intrusion Prevention System Design", Proceedings of the Fourth International Conference on Computer and Information Technology, 2004. 9.

[2] Snort. http://www.snort.org/

[3] A. Aho, M. Corasick, “Efficient string matching: an aid to biliographic search”, Comm. ACM. 18:333-40, 1975.

[4] J. Lockwood, "Fast and Scalable Pattern Matching for Content Filtering", Architectures for Networking and Communication System(ANCS), 2005. 10

[5] 전용희, “침입방지시스템(IPS)의 기술 분석 및 성능평가 방안”, 정보보호학회지, 제 15권, 제 2호, 2005. 4.

[6] An NSS Group Report V 1.0, "Intrusion Prevention Systems(IPS), Group Test", NSS, 2004. 1.

[7] Smartbits, http://www.spirentcom.com/