O/S 정보를 이용한 침입탐지 처리성능 향상에 관한 시스템 설계 및 구현

손만경* · 이동휘* · 김귀남*

요 약

네트워크의 속도가 빨라지고 인터넷의 보편화로 인하여 웜, 이메일 바이러스 등 악의적인 공격이 급증 하였으며. 네트워크의 악의적인 공격에 대한 방어로 기존의 방화벽을 비롯하여 최근 침입방지시스템에 이르기 까지 수많은 방어기법이 생겨났다. 또한 악의적인 공격의 형태가 바뀜과 동시에 방어의 기법도 달라지게 된다. 가장 대표적인 방어 기법으로 snort를 들 수 있으며 공격형태가 바뀜에 따라 Snort의 Rules 파일이 증가하게 된다. 따라서 탐지수행능력이 점점 떨어지게 된다. 본 논문에서는 Snort의 Rule 파일을 O/S별로 구분하여 처리성능 향상을 위한 구조를 제안하고 설계 및 구현한다. 이 시스템은 Snort의 기본 구성보다 처리성능을 향상시킬 수 있다.

Designing and Realization of the System for the Improvement of Processing Capability of Intrusion Detection by Using O/S Information

Man Kyung Son* · Dong Hwi Lee* · Kuinam J Kim*

ABSTRACT

As the speed of network has fastened and the Internet has became common, an ill-intentioned aggression, such as worm and E-mail virus rapidly increased. So that there too many defenses created the recent Intrusion detection system as well as the Intrusion Prevention Systems to defense the malicious aggression to the network. Also as the form of malicious aggression has changed, at the same time the method of defense has changed. There is "snort" the most representive method of defense and its Rules file increases due to the change of aggression form. This causes decline of capability for detection. This paper suggest, design, and realize the structure for the improvement of processing capability by separating the files of Snort Rule according to o/s. This system show more improvement of the processing capability than the existing composion.

Key words : Snort, IDS, IPS

* 경기대학교 정보보호학과

1. 서 론

전 세계적으로 P2P 소리바다 와 같은 멀티미디어의 전송이 활발해 지고 더욱 다양해지면서 네트워크의 트래픽 급증을 가져왔다. 네트워크의 이용 확대를 위해 이미 수 기가비트급 망이 확장 됐으며 이러한 네트워크 속도 및 대역폭의 증가추세는 기가비트 이더넷과 VDSL과 같은 고속 가정용 인터넷의 보편화로 더욱 가속화 되고 있다. 인터넷의 확장과 더불어 네트워크 속도의 증가로 네트워크를 통한 서비스 거부 공격, 인터넷 웜, 이메일 바이러스 등의 악의적인 공격도 증가하고 이로 인해 피해도 급격히 증가하는 추세이다. 예를 들면 Code Red Worm[7]이나 SQL Slammer Worm[8]은 수시간 또는 수분만에 막대한 피해를 입혔다.

네트워크 시스템에 대한 악의적인 공격에 대한 방어로 기존에는 방화벽과 네트워크 침입탐지 시스템에 의존하였다. 방화벽은 특정 포트로 들어오는 공격은 막을 수 있으나 공개된 포트를 사용하는 경우의 공격에는 대응할 수 없는 취약점을 갖고 있다. NIDS의 Snort는 이러한 취약점을 보완하여 패킷의 헤더 뿐만 아니라 데이터도 조사함으로서 공격이 발생 했는지를 탐지할 수 있다. 하지만 네트워크 주 경로 옆에 설치되어 들어오는 패킷을 직접 막지 않고 통과시키기 때문에 침입이 발견되었을 때에는 공격하는 패킷은 이미 호스트에 도달 된 뒤 일수 있다는 문제점이 있다. 따라서 이런 문제점을 보완하고 능동적으로 악의적인 공격을 방지하는 네크워크 침입방지시스템이 등장하였다.

Snort는 패킷 데이터 조사에는 이미 알려진 공격패턴을 패킷의 데이터에서 패턴매칭을 통해 찾는다. 이 패턴매칭 과정은 침입을 탐지하는 과정 중에서도 가장 많은 계산을 요구한다. 예를 들면 NIDS이며 인라인 모드로 동작하여 침입탐지기능을 수행 할 수 있는 Snort에서 패턴매칭이 수행되는 부분은 전체 수행에서 70~80%의 부분을 차지한다. Snort는 streem matching 방법으로서 패킷이 들어오면 Snort의 rule 첫 번째부터 순차적으로 마지막 rule까지 패턴매칭을 시도한 후 빠져 나간다. 따라서 rule의 개수가 많아질수록 탐지수행속도는 떨어지기 마련이다.

본 논문에서는 기존 Snort의 패턴매칭 방법에 O/S Plug-In을 추가하여 패킷필터링 탐지 수행능력 향상에 관한 방법을 제시한다. 2장에서는 침입탐지 시스템의 기본탐지 방법에 Boyer-Moore 알고리즘과 Aho-corasick 알고리즘에 대해 알아본다. 3장에서는 Snort의 기본구성에 대해 설명한다. 4장에서는 Snort의 기본구성에 O/S Plug-In을 추가하여 처리속도 향상에 관한 구조를 설명하고 평가한다. 마지막으로 5장에서 논문의 결론을 맺는다.

2. 침입탐지 시스템의 기본 탐지 방법

2.1 Boyer-Moore 알고리즘

Boyer-Moore 알고리즘의 기본 방식은 문자를 뒤에서부터 비교 하여 맨 앞의 스트링 문자까지 맞게 되면, 검색이 성공했음을 알린다. 불일치 문자 발생시에 기본적으로 '불일치 문자 방책(Bad Character Shift)'이라는 방식을 쓰게 되는데, 이에 단점이 있어 일치 '접미부(Good Suffix Shift) 방식'과 혼용하여 성능 향상을 꾀하였다. '불일치 문자 방책'이란 불일치 문자 발생 시 불일치가 일어난 곳의 텍스트 문자를 스트링에서 검색하여 가장 최근에 발생한 위치의 다음번의 스트링 문자부터 텍스트와 검색하는 방식으로 Boyer-Moore 알고리즘에서 기본적으로 사용되는 정책이다[1].

Boyer-Moore 알고리즘은 Snort v1.6에서 기본으로 사용하는 방법으로. 규칙의 구성은 RTN(Rule Tree Nodes)에 표시되며, 옵션의 구성은 OTN (Option Tree Nodes)에 표시된다. RTN에는 여러 규칙이 공통적으로 가질 수 있는 조건들을 닮고

있다. OTN은 각 규칙에 더해질 수 있는 옵션들을 담고 있다. 예를 들면 RTN은 출발지 주소(source address) 또는 목적지 주소(destination address) 그리고 출발지 포트(source port) 또는 목적지 포트(destination port)로 구성된다. 또한 TCP, ICMP, UDP등의 프로토콜 타입이 있다. OTN은 TCP flag 또는 ICMP code와 type, 패킷내용(payload) 크기와 패킷 내용 등 각 규칙이 추가되어야 할 다양한 옵션에 대한 정보를 포함한다. 이중, 패킷 내용을 검사하는 부분이 침입탐지 시스템의 패킷 처리수행에 있어서 병목이 되는 부분이다.

이러한 구조는 체인 형태로 구성되는데 체인의 헤더 부분에는 RTN이 왼쪽에서 오른쪽으로 연결되어 있고 OTN은 각 RTN 아래에 연결되어 있다.

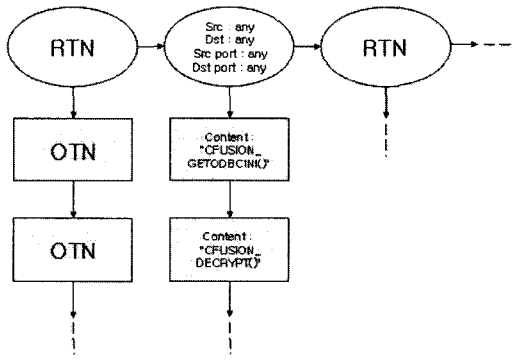
RTN-OTN 방법에서 패킷을 탐지할 때에는 우선 RTN 목록을 왼쪽부터 오른쪽으로 검사해 나간다. 현재의 패킷 내용이 RTN 내용과 일치하는 경우 해당 RTN 아래에 있는 OTN 목록을 위에서 아래로 검사해 나간다. 각 OTN 목록에 저장된 옵션 정보들은 플러그인 함수를 이용하여 검사되는데, 하나의 OTN 내에 여러 개의 플러그인 함수가 있는 경우 이들은 연결리스트(Linked list)로 구성된다. 플러그인 함수 중 하나라도 실패하면 그 OTN에 대한 검사는 실패한 것이며 이 경우 다음 OTN 검사를 시작한다.

Snort v1.6의 경우, 패킷의 내용을 검사하는 것이 성능의 병목 현상이 된다는 가정 하에 다른 OTN에 있는 모든 옵션들 까지 검사하고 난 이후에 패킷 내용을 검사하게 된다. 패킷 내용 검사에는 Boyer-Moore 알고리즘을 사용하는데 한 OTN 검사가 끝나면 그 다음 OTN에 저장된 패킷 내용을 검사하게 된다.

이 때, 한 OTN에는 패킷 내용 중에 있는 scripts/hello.exe 문자열을 검사하라는 지정이 있고 다른 OTN에는 패킷 내용 중에 있는 scripts/hi.exe 문자열을 검사하라는 지정이 있는 경우를 생각해 보면 첫째는 패킷 전체 내용에 대하여 scripts/hello.exe 문자열을 찾아보고 이것이 실패하면 다음으로 패킷 전체의 내용에 대하여 두 번째 문자열인 scripts/hi.exe 문자열을 찾는다. 만약, 검사하고자 하는 패킷에 scripts 라는 문자열이 없는 경우, 첫 번째 검사시 scripts가 없다는 것을 알아서 두 번째 검사내용인 scripts/hi.exe 를 검사하는 것은 실패할 것이 분명함에도 Boyer-Moore 알고리즘을 사용하는 경우 두번째 검사를 패킷 내용 전체에 대하여 진행하며 이로 인해 성능의 저하를 야기 시킨다[2, 3].

```
Alert TCP any any    > any any
(content : "CFUSION_GETODBCINI()";)
Alert TCP any any    > any any
(content : "CFUSION_DECRYPT()";)
Alert TCP any any    > any any
content : "CFUSION_SETODBCINI()";)
Alert TCP any any    > any any
(content : "CFUSION_SETTINGS()";)
```

(그림 1) Snort의 WEB-COLDFUSION Rules



(그림 2) Boyer-Moore 알고리즘의 Rule 구성

Snort ver1.6에서 패킷 내용 검사에 Boyer-Moore 알고리즘을 사용하는 구체적인 예는 다음과 같다. (그림 1)과 같은 Rule이 있을 때 Boyer-Moore 알고리즘은 이 Rule을 RTN/OTN chain으로 처리하게 되는데 이것이 (그림 2)에 표현되고 있다.

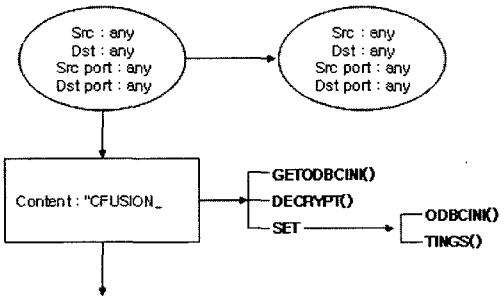
Boyer-Moore 알고리즘은 패킷 내용 탐색에

OTN당 한 번식, 총 4번을 수행한다. 만약 패킷의 어디에도 접두사 "CFUSION"이 존재하지 않는다고 가정하면 패킷의 첫 번째 내용 검사가 실패할 것이고 나머지 OTN에 대한 내용 검사도 실패가 예상되지만 Boyer-Moore 알고리즘은 단일패턴을 탐색하므로 실패가 예상됨에도 계속 검사를 진행하게 된다. 따라서 하지 않아도 될 검사를 함으로써 패턴 매칭 처리속도가 떨어지게 된다. 이것을 개선한 방법이 Aho-corasick 알고리즘을 사용한 다중 키워드 방식인 AC-BM(Aho-corasick Boyer-Moore)방식이다.

2.2 Aho-Corasick 알고리즘

Snort v2.0 이하에서 사용되던 Boyer-Moore 알고리즘의 구조를 변경하여 패킷 매칭 속도를 크게 향상시킨 방법이다. Aho-corasick 방법은 Boyer-Moor 방식의 패킷 내용 탐색을 비교해야 할 키워드 별로 매번 진행해야 하는 번거로움이 있는 점을 개선한 방법이다[3].

Aho-corasick 방법은 한 RTN에 소속된 OTN들 내에 있는 검사해야 할 키워드들에 대하여 패킷 내용 탐색을 한꺼번에 진행한다(그림 3). 이 때 사용되는 문자열 비교(String matching)알고리즘은 Boyer-Moore 알고리즘을 다중 키워드 검색으로 확장시킨 AC-BM(Aho-corasick Boyer-Moore) 방법이다[4].



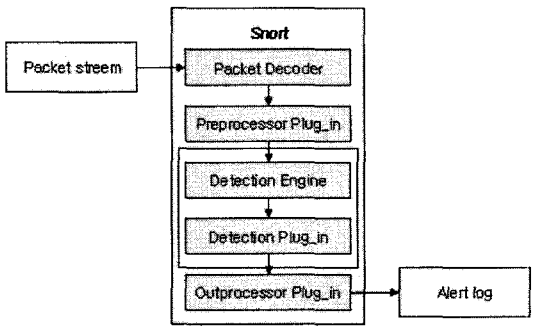
(그림 3) AC-BM 알고리즘의 Rule 구성

Snort v2.0에서는 규칙들이 프로토콜과 출발지와 목적지의 포트번호로 규칙들이 집단(Ruleset)을 이루고 다중 패턴 매칭 기법을 사용하여 규칙과 일치하는 패킷을 찾는 과정의 속도를 향상시켰다. 패킷이 도착하면 패킷의 프로토콜과 출발지와 목적지 포트번호를 가지고 알맞은 규칙집단을 찾고 그 규칙집단에 있는 모든 'content' 옵션의 문자열 패턴에 대하여 다중 패턴 매칭 방법을 사용하여 조사한다. 만약 매칭 되는 패턴이 발견되면 그 패턴을 갖고 있는 모든 규칙에 대하여 RTN과 OTN이 차례로 조사된다. 만약 패턴이 발견되지 않으면 그 패킷은 'content' 옵션을 갖는 모든 규칙에 대하여 조사를 한다. 이 방법에서는 패턴이 일치하지 않는 'content' 옵션을 갖는 모든 OTN을 조사에서 배제함으로서 Snort 패킷 매칭속도를 월등히 향상시켰다[5].

이러한 다중 패턴 매칭(AC-BM)방법은 Snort의 성능에 큰 영향을 미치는데 주로 사용되었고 Snort v2.0 이후로 계속 사용되고 있다.

2.3 Snort 구성요소

Snort는 서명-기반 분석 기능이 추가되면서 부터 비로소 Snort는 침입 탐지 시스템으로 발전하기 시작하였다. Snort v1.5부터 현재의 Snort 구조의 근간이 되는 플러그-인 구조를 사용하기 시작



(그림 4) Snort 구성요소

했고, 2003년 4월 Snort v2.0이 나오게 되었다. 현재 최신 버전인 2.4은 약 75,000 줄의 코드로 작성되어 있다[5].

Snort는 스니퍼(sniffer), 전처리기(preprocessor), 탐지엔진(detection Engine), 출력모듈(outprocessor)의 4가지 구성 요소로 이뤄져 있다(그림 4).

2.3.1 스니퍼(sniffer)

스니퍼(sniffer) 또는 패킷 스니퍼는 네트워크에서 데이터를 수집하는 소프트웨어나 하드웨어를 의미한다. Snort는 원래 스니퍼 용도로 만든 프로그램이기 때문에 기본적으로 네트워크를 돌아다니는 패킷을 잡아낼 수 있다.

전처리기(preprocessor)의 전단계로서 패킷을 재조합한다. 대표적인 도구로서 tcpdump를 들 수 있다.

2.3.2 preprocessor

스니퍼에서 캡처한 네트워크 패킷은 전처리기(preprocessor)로 이동한다. Snort의 전처리기는 패킷을 탐지 엔진에서 비교하기 전에 사전 처리 작업을 해 주는 역할을 한다. 예를 들어, stream 전처리기는 여러 TCP 패킷을 하나로 모음으로써 여러 패킷에 걸친 공격을 잡아내는 역할을 한다. Snort의 다른 부분과 마찬가지로 전처리기도 플러그-인 방식으로 되어 있다. 플러그-인 방식의 장점은 그때그때 필요한 플러그-인을 추가하거나 삭제하기가 쉽기 때문에 훨씬 유연한 소프트웨어를 운영할 수 있다는데 있다.

2.3.3 탐지 엔진

Snort의 핵심 모듈로서 패킷과 규칙을 비교하여, 패킷에 해당하는 규칙이 있을 경우 경고를 발생한다. Snort의 규칙은 Snort에서 가장 중요한 부분으로 Snort에는 나름대로 규칙 문법이 있다. 규칙 문법은 프로토콜의 종류, 컨텐트, 길이, 헤더, 기타 여러 요소(버퍼 오버플로우를 정의하기 위한 쓰레기 문자 등)를 포함하고 있다. 규칙을 잘 설정하면 Snort를 자신의 환경에 맞게 커스터마이징하는 것이 가능하다.

2.3.4 출력 모듈

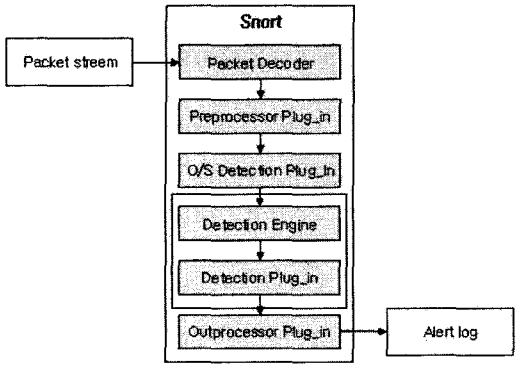
Snort가 발생시킨 경고는 출력 모듈로 전송된다. 출력 모듈도 플러그-인 구조로 되어 있으며, 현재 Snort는 로그 파일, 네트워크 연결, UNIX 소켓 또는 윈도우 팝업(SMB), 또는 SNMP 트랩, MySQL과 Postgres와 같은 SQL 데이터베이스로 경고를 보낼 수 있다. 그리고 Snort의 경고를 분석하기 위한 여러 툴이 나와 있다. 그 중 대표적인 툴로 SnortSnarf와 Swatch, ACID 등을 들 수 있다.

3. 침입탐지 성능 향상을 위한 Snort 설계 및 구현

Snort 서버는 주기적으로 Nmap-O 옵션으로 네트워크 호스트에 대한 O/S 정보를 획득한다. 이 정보는 crond 데몬에 의하여 주기적으로 탐색을 하며 userinfo.conf 파일에 기록된다. 기록된 userinfo.conf 파일은 패킷이 들어오면 Snort의 어떤 signature로 들어갈 것인가의 방향을 결정 짓는다.

이렇게 통과된 패킷은 이미 정해진 Snort signature로 들어가게 되며 O/S의 정해진 패턴만을 탐색하게 된다. 따라서 자신의 O/S와 관계가 없는 signature는 탐색을 하지 않게 되므로 그만큼 처리성능이 향상 된다.

본 논문에서 제안하고자 하는 내용은 (그림 4) Snort의 기본 구조에서 O/S Plug-In 정보가 추가되며 Snort Rules signature 파일을 분리하여 실제 호스트에 치명적인 패턴만 탐지하게 함으로써 처리성능을 향상시키는데 있다. (그림 5)는 기존의 Snort 처리방식에 O/S 탐지 플러그인이 추가되어 O/S를 탐지하고 이를 참조하여 탐지엔진으로 보내는 것을 보여주는 전체 구성도이다.



(그림 5) 제안한 전체적인 구성도

3.1 Nmap을 이용한 OS탐지

일반적으로 TCP/IP fingerprinting은 "-O" 옵션을 포함하여 원격으로 운영체제를 탐지해낸다. 이것은 ping scan을 제외한 port scan과 결합하여 사용해야만 한다. Nmap은 호스트에 다른 타입의 조사를 행하여 O/S를 찾아낸다. 원격으로 O/S를 탐지해내기 위한 다른 방법과 마찬가지로 ICP initial Sequence Number(ISN)의 패턴을 찾기 위해 SYN packet과 함께 선언하지 않은 flag를 리모트 호스트로 보내고, BOGUS flag는 원격 호스트의 그 반응이 어떤 종류인가를 입증하기 위해 FIN 조사같은 기능을 사용한다. 그것은 TCP stack에서 포함하고 있고 이를 Fingerprinting하게 된다[6].

3.2 crontab를 이용한 주기적인 O/S탐지

Userinfo.conf에 저장된 각 내부 호스트의 O/S 정보들을 주기적으로 갱신시켜 주기 위하여 Crontab의 주기작업을 이용한다. 탐지대상 호스트가 추가되거나 사라지면 userinfo.conf에서 삭제하거나 추가하기 위하여 주가작업을 실행한다. 이와 같은 작업이 이루어 지지 않을 경우 한번 등록된 정보를 계속 사용하게 되기 때문이다. 또한 직접 관리자가 입력하거나 삭제 하는 데는 많은 시간과 노력이 필요하기 때문이다. 이 주기작업은 관리자에 의해 주기 시간이 결정되며 트래픽이 많이 발생하지 않는 새벽시간 이루어 져야 한다.

3.3 탐지된 O/S 정보저장

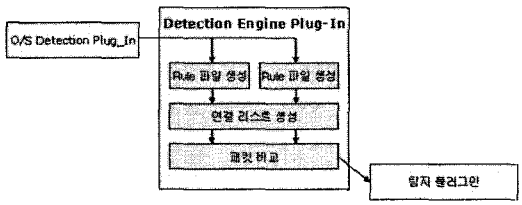
Nmap를 이용하여 획득한 O/S 정보는 userinfo.conf 파일에 저장된다. 이 userinfo.conf 파일은 Nmap 스캐닝으로 인하여 cron 데몬이 동작할 때 주기적으로 갱신된다.

기존의 Snort 구성요소에서 O/S 정보를 획득하여 저장하고 저장된 정보를 이용하여 실제 대상 호스트에 치명적인 패킷패턴만 감시하기 때문에 기존의 전체 Rules 파일 검색 시간보다 빠르게 진행 할 수 있다.

패킷을 분석 및 탐지하기 위한 과정의 일부분으로 O/S를 탐지하기 위한 절차는 위와 같은 방법으로 인하여 진행되고 수집한다. 이렇게 수집된 정보는 Snort signature 정보로 들어가게 되어 검사를 수행하게 된다.

3.4 Snort Rules Signature 구조

기존의 단일구성 Snort signature에서 다음 (그림 6)과 같이 Rules 파일을 분리한다. 이것은 실제 호스트의 O/S 정보에 따라 패킷패턴을 각각 다른 Rules 파일을 거치게 함으로서 기존의 처리되는 패턴숫자보다 적게 하기 위해서이다.



(그림 6) Rules 파일 분리

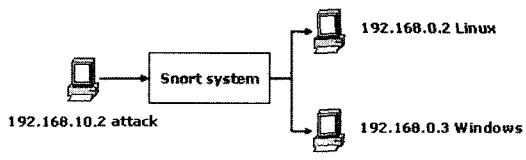
예를 들면 기존의 Rules 파일에는 약 3500개의 탐지룰이 있다. 패킷이 들어오면 Snort는 이 3500

개의 룰을 순차적으로 탐지해 나간다. 최악의 상황에서 3500 번째에 룰에서 패킷매칭이 이루어 졌다면 앞의 룰 탐색에서는 하지 않아도 될 탐색이 이루어 진 것이 분명하다. 따라서 그만큼 속도에 영향을 줄 것이다.

O/S 별로 취약점이 있다는 것은 Snort도 알고 있다. 그러나 알고 있음에도 해당 O/S와 관련이 없는 Rule까지 탐지한다. 다른 관점에서 볼 때 분명히 비효율적이다. 다시한번 예를 들면 Snort가 크게 O/S별로 linux패턴 rules 1000개의 파일들과 windows rules 2500개의 파일이 각각 분리되어 있다. 실제대상 호스트가 192.168.0.2이고 linux 서버라고 가정해 보자. 이 서버가 가지고 있는 취약점이 SSH2의 취약점을 이용한 공격 이며 이 취약점은 다른 O/S에는 없다라고 가정한다. 이때 패킷이 들어오면 userinfo.conf에 저장 되어있는 목적지 주소와 O/S 종류만 참조하여 Linus rules로 갈 것인지 windows rules로 갈 것인지가 결정된다. 위의 가정과 같은 경우 linux rules 파일로 들어갈 것이다. 따라서 Snort는 최악의 경우 linux 패턴룰 1000개만 탐색하면 되고 나머지 windows 패턴룰 2500개는 탐색를 하지 않아도 되므로 그만큼 탐색하는 시간이 단축되게 된다.

3.5 성능 측정 및 결과

성능측정은 기본 상태의 Snort 시스템에서 처리하는 Snort 수행시간과 본 논문에서 제한하는 Snort 시스템의 수행시간에 대하여 측정 하였다. Snort 엔진을 사용하는 linux시스템 커널 버전은 2.4.20-br25b2이며 Snort 엔진버전은 2.1.1이다.



(그림 7) 성능시험 구성도

성능 측정을 위하여 (그림 7)과 같이 구성하였으며 다음과 같은 절차를 거친다.

1) 기존 Snort 시스템의 Snort 수행시간 측정
2) Snort 시스템 설계 후 linux 서버로 공격했을 때 Snort 수행시간 측정
3) Snort 시스템 설계 후 windows 서버로 공격했을 때 Snort 수행시간 측정

측정은 5회 평균값을 계산하여 적용하였다.

측정을 위한 공격으로 Nmap을 이용한 Port scan 이용하였으며 Snort 수행시간을 알기 위한 방법으로 time 명령을 이용하여 Snort를 실행시킨다.

1) nmap 명령어
   nmap -sS -PT -PI -O -T 3 192.168.0.2(바뀜)
2) Snort 명령어
   % time /sbin/snort/snort-lib

Snort의 CPU 사용시간을 알기 위하여 위의 명령어 실행 후 공격을 하고 Ctrl+C를 눌러 time를 종료시킨다.

위와 같이 측정한 결과 다음과 같은 결과가 나타났다.

〈표 1〉 테스트 결과

		linux	windows
분리전	경과시간	50.635	45.258
	CPU사용	0.154	0.148
분리후	경과시간	48.259	59.254
	CPU사용	0.145	0.137

경과 시간은 실제 Snort 데몬이 올라기 있는 상태의 시간을 나타내며 CPU 사용시간이 Snort가 패킷을 비교 분석에 사용한 시간이다.

위 <표 1>의 결과를 볼 때 기존의 Snort 패킷

필터링의 속도보다 제안한 속도가 더 향상 됐음을 확인할 수 있다.

4. 결 론

본 논문에서는 Snort signature Ruleset을 O/S별로 분리하여 패턴매칭 속도의 향상을 위한 방법이 제안되었다. 이를 위해 Snort가 현재 사용되고 있는 알고리즘 및 Snort의 구성요소에 대하여 기술하였으며 이 구성요소에 O/S를 탐지하는 Plug-In을 설계하여 기존의 Snort보다 탐색하는 시간을 단축시키는 방법에 대해 각각의 기능과 역할에 대하여 정의하였고 이를 바탕으로 Snort 시스템을 설계 및 구현하였다.

기반 기술로는 Snort의 Boyer-Moore 알고리즘과 Aho-corasick 알고리즘에 대해 소개했으며 본 논문에서 제안되는 시스템의 바탕인 Snort의 기본 탐지 방법에 대해 자세하게 기술하였다. 특히 Snort가 패턴매칭을 통하여 탐지하는 과정을 자세히 기술하였으며 이를 통해 시스템을 설계하였다. 설계를 바탕으로 구현된 시스템에서 기존의 Snort 구성과 제안된 구성으로 각각 처리성능을 비교해본 결과 처리성능이 향상됐음을 검증하였다.

제안된 Snort 시스템은 탐색하지 않아도 될 부분을 배제하여 O/S의 구분에 의해 기존의 Snort 시스템보다 탐지성능이 빠르게 나타났다.

향후 연구 과제로는 본 논문에서 제외하였던 O/S별로 공통적인 취약점들이 있는 경우와 그리고 내부에서 외부로 패킷이 흐르는 경우에 Snort signature를 어떻게 처리할 것인가에 관하여 연구가 남아 있다. 따라서 공통적인 취약점이 있는 경우 Snort rules 파일의 구분과 처리방법이 진행되어야 할 것이며 Snort를 기준으로 내부에서 외부로 흐르는 패킷은 어떤 프로세스로 처리할 것인가에 대한 연구가 남아있다.

참 고 문 헌

[1] R. COLE, "Tight Bounds on the Complexity of the Boyer-Moore Pattern Matching Algorithm", SIAM Journal on Computing, Vol. 23, No. 5, pp. 1075-1091, 1994.

[2] M. Fisk from Los Alamos National Laboratory, G. Varghese from University of California Sandiego, "Applying Fast String Matching to Intursion Dection."

[3] C. Coit, S. Staniford, and J. McAlerney, "To Faster String Matching for In trusion Detecton, "Proceedings of DARPA Information Survivability Conference and Exposition (DISEC II)", Anaheim, California, June 12 Vol. 14, 2001.

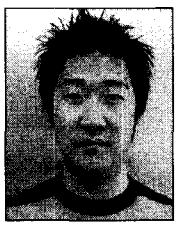
[4] A. V. Aho and M. J. Corasick, "Efficient String Matching : An Aid to Bibliographic Search", Communications of the ACM, Vol. 18, No. 6, p. 333, p. 340, 1975.

[5] Snort site. http://www.snort.org

[6] http://www.insecure.org/nmap/nmap/finger printing/article.html

[7] Code Red worm exploiting buffer overflow in IIS indexing service DLL. CERT advisory CA-2001-19, jan 2002.

[8] MS-SQL Server Worm. CERT advisory CA-2003-04, jan 2003.



손 만 경

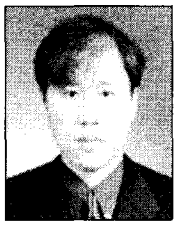
2000년 공주산업대학교 산림자원학과(이학사)

2005년 경기대학교 정보보호 기술공학과(공학석사)

2003~현재 (주)시큐어넥서스 연구원

이 동 휘

2000년 경기대학교 전자계산학과(이학사)
2003년 경기대학교 정보보호기술공학과(공학석사)
2004~ 현재 경기대학교 정보보호기술공학과 박사과정



김 귀 남

미국 캔자스대학 수학과(응용수학사)
미국 콜로라도주립대학 통계학과(통계학석사)
미국 콜로라도주립대학 기계산업공학과(기계·산업공학박사)
현재 경기대학교 정보보호학과 주임교수