ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

Snort 2.9.0 환경을 위한 TCAM 기반 점핑 윈도우 알고리즘의 성능 분석

Performance Analysis of TCAM-based Jumping Window Algorithm for Snort 2.9.0

이 성 윤* 류 기 열**
Sung-Yun Lee Ki-Yeol Ryu

요 약

스마트 폰 이용자의 급격한 증가에 따른 무선 네트워크의 지원 및 모바일 환경은 언제 어디서나 네트워크를 이용할 수 있게 되었다. 이러한 인터넷 망의 발달로 인해 네트워크 트래픽이 급증함으로써 네트워크를 통한 분산서비스 공격, 인터넷 웜, 이메일 바이러스 등의 다양한 악의적인 공격이 증가되고 이에 따른 패턴이 급격하게 증가하는 추세이다. 기존 연구에서 침입탐지시스템인 Snort 2.1.0 룰의 약 2,000개 패턴으로 M-바이트 점핑 윈도우 알고리즘을 적용한 결과를 분석하였다. 하지만 점핑 윈도우 알고리즘은 패턴의 길이와 수에 큰 영향을 받기 때문에 더 긴 패턴과 더 많은 패턴을 갖는 새로운 환경(Snort 2.9.0)에서 TCAM 룩업 횟수와 TCAM 메모리 크기에 대한 새로운 분석이 필요하다. 이 논문에서는 Snort-2.9.0 룰에서 약 8,100개의 패턴을 이용하여 윈도우 크기별 TCAM 룩업 횟수와 TCAM의 크기를 시뮬레이션 했고 그 결과를 분석하였다. Snort 2.1.0에서는 16-바이트 윈도우에서 9Mb의 TCAM이 최적을 효과를 낼 수 있는 반면, Snort 2.9.0에서는 16-바이트 윈도우에서 18Mb TCAM 4개를 캐스케이딩으로 연결할 경우 최적의 효과를 낼 수 있다.

ABSTRACT

Wireless network support and extended mobile network environment with exponential growth of smart phone users allow us to utilize the network anytime or anywhere. Malicious attacks such as distributed DOS, internet worm, e-mail virus and so on through high-speed networks increase and the number of patterns is dramatically increasing accordingly by increasing network traffic due to this internet technology development. To detect the patterns in intrusion detection systems, an existing research proposed an efficient algorithm called the jumping window algorithm and analyzed approximately 2,000 patterns in Snort 2.1.0, the most famous intrusion detection system. using the algorithm. However, it is inappropriate from the number of TCAM lookups and TCAM memory efficiency to use the result proposed in the research in current environment (Snort 2.9.0) that has longer patterns and a lot of patterns because the jumping window algorithm is affected by the number of patterns and pattern length. In this paper, we simulate the number of TCAM lookups and the required TCAM size in the jumping window with approximately 8,100 patterns from Snort-2.9.0 rules, and then analyse the simulation result. While Snort 2.1.0 requires 16-byte window and 9Mb TCAM size to show the most effective performance as proposed in the previous research, in this paper we suggest 16-byte window and 4 18Mb-TCAMs which are cascaded in Snort 2.9.0 environment.

☞ keyword : 분산서비스거부(Distributed Denial of Service), 침입탐지시스템(Intrusion Detection System), 점핑윈도우알고리즘(Jumping Window Algorithm), Snort2.9.0, TCAM

1. 서 론

최근 인터넷의 상용화 수준과 망의 속도가 급증함에 따라, 스마트 폰의 확산과 그에 관련한 무선 네트워크의 지원, 그리고 유비쿼터스 환경은 언제 어디서나 네트워크를 이용할 수 있게 되었다. 하지만 인터넷의 확장과 네트워크 속도의 증가로 네트워크를 통한 분산 서비스 공격(DDoS), 인터넷 웜, 이메일 바이러스 등의 악의적인 공격도 빨라지고 급속히 증가되고 있으며, 이로 인한 피해도 급격히 증가되는 추세이다. 네트워크의 고속화를 바탕으로 대용량 데이터의 송수신은 보안 시스템의 처리 속도가 네트워크에 부하를 주지 않고, 완벽한 보안 기능을 지원할 수 있기 위해서 고성능을 요구하는 침입 탐지 시스템(Intrusion Detection System)에 대한 연구가 필요하다[1,2].

네트워크 침입 탐지 시스템은 악의적인 공격을 감지해내고, 인터넷 시스템을 보호하기 위해 사용된다. 네트

* 정 회 원 : 퓨처시스템 정보통신연구소 전임연구원
hwauni@naver.com
** 정 회 원 : 아주대학교 정보컴퓨터공학부 교수 (교신저자)
kryu@ajou.ac.kr
[2011/07/01 투고 - 2011/07/11 심사(2011/10/14 2차) - 2011/12/05 심사완료]

http://dx.doi.org/10.7472/jksii.2012.13.2.41

워크 침입 시도나 공격은 특정 패턴으로 표현될 수 있기 때문에 패턴 매칭이 네트워크 침입 탐지 시스템에서 가장 중요한 이슈 중의 하나이며, 사용하는 패턴 매칭 알고리즘에 따라 네트워크 침입 탐지 시스템의 성능이 결정된다. 현재 대표적인 네트워크 침입탐지 시스템인 Snort의 패턴 데이터베이스에는 수천 개의 패턴들이 등록 되어 있으며, 패턴의 수는 점점 더 늘어나고 있어 패턴 매칭에서의 병목 현상이 더욱 심해질 것으로 예상된다[3].

따라서 고속 네트워크에서 침입을 탐지하기 위해서는 하드웨어 기반 접근 방식이 필요하다[4]. Ternary Contents Addressable Memory(TCAM)는 하드웨어적으로 병렬 검색을 지원하므로 일반 메모리에 비해 빠른 검색을 수행할 수 있어 고성능 네트워크에서 효과적이고 빠르게 패턴 매칭을 수행 하도록 해준다[5,6]. 기존 연구 [7]에서는 TCAM 기반 패턴 매칭 기법인 점핑 윈도우 패턴 매칭(Jumping Window Pattern Matching)기법을 제안하고, Snort 2.1.0 룰로부터 2,394개의 패턴(평균길이 9바이트)을 수집하여 페이로드의 크기에 대한 평균 룩업시간과 윈도우 크기에 대한 TCAM 메모리의 크기를 분석하였다.

위 방식에서 처리 속도에 영향을 주는 것은 Snort 룰의 패턴 수와 패턴의 길이이다. 하지만 최근 버전인 Snort 2.9.0에서는 패턴의 수가 약 8,100개 정도이며 'content' 옵션의 평균 패턴 길이는 42-바이트로 기존의 Snort 2.1.0에 비해 약 4배 이상 패턴의 길이가 길며 또한 그 수도 많아졌다. 이러한 패턴의 길이는 늘어나고 패턴의 수는 점차적으로 증가하고 있으므로 현재의 환경에서 점핑 윈도우 알고리즘의 성능 분석을 할 필요가 있다.

본 논문에서는 Snort 2.9.0 환경에서 점핑 윈도우 패턴 매칭 기법을 적용하여 가장 적합한 TCAM 메모리의 크기와 윈도우 사이즈와의 상관관계를 분석한다. 이를 위해 점핑 윈도우 패턴 매칭 시뮬레이터를 개발하고 시뮬레이션 환경을 구축한다. 시뮬레이션 환경에서는 IDS 테스팅 프로그램인 IDSwakeup를 이용하여 패킷을 생성한 후 와이어샤크 프로그램을 이용하여 모든 패킷을 수집하여 시뮬레이션을 위한 페이로드를 생성한다. Snort 2.9.0의 패턴을 TCAM에 저장한 후 페이로드 생성기로부터 생성한 페이로드 패킷을 입력하여 점핑 윈도우 패턴 매칭을 수행한다. 윈도우 사이즈를 4~32 범위에서 변화시키면서 TCAM의 룩업 횟수와 TCAM 사이즈의 상관관계를 분석한다. 이 분석결과는 메모리의 효율성과 처리속도의 측면을 고려하여 윈도우 사이즈를 조절함으로써 침입탐지 시스템을 적용할 환경이나 네트워크 규모를 고려하여 최적의 성능을 낼 수 있는 조합을 찾는데 활용할 수 있다.

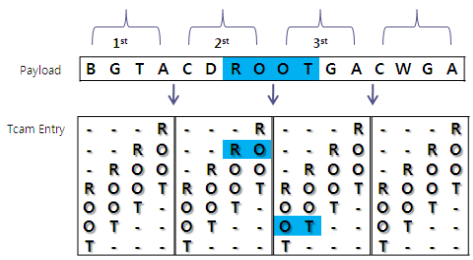
본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 TCAM 기반의 점핑 윈도우 패턴 매칭 알고리즘에 대해 살펴보고, IDS 테스팅 프로그램인 IDSwakeup에 대해서 설명한다. 3장에서는 성능분석을 위한 점핑 윈도우 패턴 매칭 시뮬레이터와 패킷 페이로드 생성기를 포함한 시뮬레이션 환경의 구축에 대하여 기술한다. 제4장에서는 Snort 2.9.0의 패턴에 대한 시뮬레이션 결과를 분석하고 평가한 후, 마지막으로 제5장에서는 결론 및 향후 과제를 제시한다.

2. 관련 연구

2.1 점핑 윈도우 방식

점핑 윈도우 알고리즘[7]은 TCAM을 이용하여 패턴 매칭을 수행하는 하는 고성능 알고리즘이다. M-바이트 점핑 윈도우 패턴 매칭 방식은 M 바이트 당 한 번의 TCAM 검색을 수행하여 패턴을 찾아내는 것이다. (그림 1)은 4-바이트 점핑 윈도우 알고리즘을 이용하여 ROOT라는 패턴을 검색하는 예이다. 페이로드를 4 바이트씩 잘라 한번에 TCAM의 4-바이트 패턴과 매치시킨다. 첫 번째 매칭을 실패한 후, 두 번째 4 바이트의 매칭은 TCAM의 두 번째 엔트리(--RO)와 부분적인 매치가 일어나고 세 번째 매치에서 나머지 부분(OT--)을 매치시켜 성공하게 된다. 즉, 3번의 TCAM 검색으로 패턴을 찾아내었다. (그림 1)에서 TCAM 엔트리의 '-'는 don't care 상태로 어떤 문자와도 매치가 된다는 것을 의미한다.

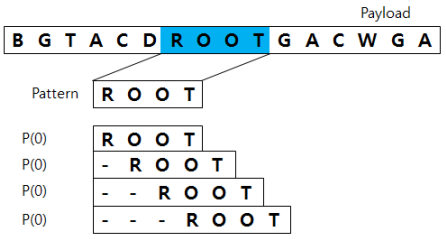
점핑 윈도우의 경우에는 첫 번째 매칭 이후에 연속되는 윈도우에 걸쳐 패턴이 매치 될 수 있으므로, 이전 매치에 대한 서브 패턴들을 검색해 낼 수 있어야 하고 서브 패턴 간의 상태를 저장하기 위한 전처리 과정이 필요하다. 전처리를 통해 패턴이 페이로드의 임의의 위치에 존재하더라도 점핑 윈도우 방식을 이용하여 패턴 매치를 수행할 수 있다.



(그림 1) 점핑 윈도우 패턴 매칭(M=4)

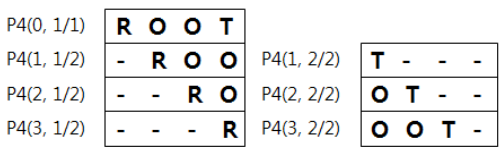
다음으로 패턴 P로부터 M-바이트 점핑 윈도우를 위한 TCAM 엔트리를 생성하는 방법을 알아보자. (그림 2)와 같이 페이로드에서 임의의 M-바이트 서브스트링을 P와 비교할 수 있도록 하기 위해 P를 0~(M-1)만큼 오른쪽으로 쉬프트한 각 패턴 (Shifted Pattern, SP)을 만들어 TCAM 엔트리 내에 저장한다.

예를 들어, 4-바이트 점핑 윈도우를 사용할 때 "ROOT" 패턴이 페이로드 상에 나타날 수 있는 위치는, (그림 2)에서처럼 패턴을 0~3만큼 오른쪽으로 시프트 한 'ROOT', '-ROOT', '- - ROOT', '- - - ROOT' 중 하나일 수 있다. 이렇게 만들어진 SP들을 TCAM 엔트리로 생성하면 페이로드에서 M 바이트 당 한 번의 TCAM 룩업을 수행할 수 있다.



(그림 2) Shift Pattern 예제

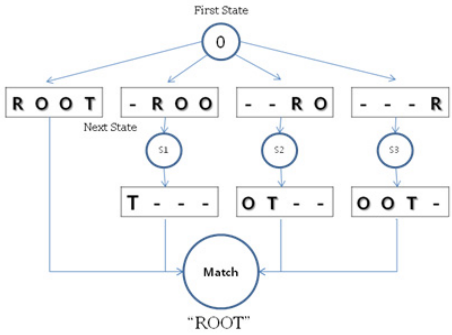
이 때, 각각의 SP들이 점핑 윈도우 크기(M)보다 커질 경우, 점핑 윈도우 크기(M) 만큼 나눈 서브 패턴(Cutting Shifted Pattern, CSP)을 TCAM 엔트리로 생성하여 저장한다. 마지막 CSP는 M-바이트 점핑 윈도우 크기를 맞추기 위해서 마지막 문자 뒤에 don't care를 채워 넣어야 한다. 예를 들면, 4-바이트 점핑 윈도우를 사용할 때 (그림 3)과 같이 'ROOT' 패턴 P는 7개의 CSP를 가진다.



(그림 3) Cutting Shifted Pattern 예제

하나의 패턴 매칭을 위해서는 동일한 패턴에서 파생한 모든 서브패턴들이 연속적으로 일치되어야 한다. 모든 서브패턴들이 연속적으로 검색되기 위해서는 그 순서에 대한 정보를 가지고 있어야 한다. 즉, TCAM 룩업으로 첫 서브패턴이 일치되면, 다음번의 TCAM 룩업을 실행할 때에는 이전에 부분 매칭된 서브패턴에 대한 정보를 가지고 있어야 한다. 이를 위해 상태 전이 다이어그램을 이용한다.

(그림 4)는 서브패턴들의 상태 전이 다이어그램의 예를 보여준다. 패턴 'ROOT'의 초기 상태는 '0'으로 설정해두고 TCAM 룩업으로 'ROOT'가 매칭되면 그 상태를 MATCH로 전이시켜 'ROOT'의 검색이 완료된다. 다른 경우로 초기상태 '0'에서 TCAM 룩업 '-ROO'가 매칭되면 S1 상태로 전이되고, S1 상태에서 다음 서브패턴 'T---'가 매칭되면 MATCH 상태로 전이되어 'ROOT'의 검색이 완료된다. 이렇게 상태전이를 이용함으로써 페이로드 내에서 연속적인 서브패턴의 검색을 수행할 수 있다.



(그림 4) 상태전이 다이어그램

2.2 트래픽 생성기 IDSwakeup

2.2.1 IDSwakeup 개요

IDSwakeup[8]은 NIDS(Network Intrusion Detection System)[9]를 테스트하기 위한 여러 공격기법들을 모아 실행하는 쉘 스크립트이다. 이 IDSwakeup 쉘 스크립트에서는 hping2와 iwu라는 일종의 공격 도구를 이용하여 (표 1)에 열거된 여러 공격기법들을 실행하는데 사용한다.

(표 1) IDSwakeup 공격 유형

IDSwakeup 공격유형	
IWU 공격	Hping2 공격
send teardrop	send icmp_bestof
send land	send dos_snork
send back_orifice	send ddos_bestof
send ping_of_death	send ftp_bestof
send finger_redirect	send telnet_bestof
send ftp_cwd_root	send rlogin_bestof
send SMBnegprot	send tcpflag_bestof

2.2.2 IDSwakeup 사용법

IDSwakeup의 기본 사용법은 아래와 같다.

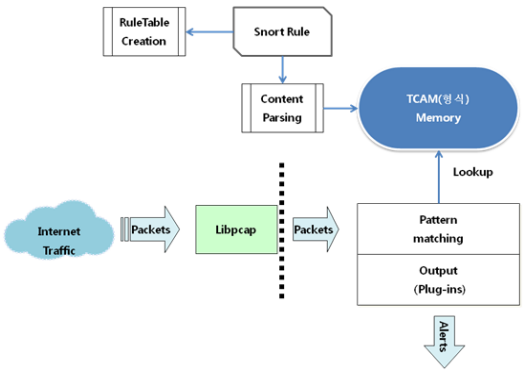
IDSwakeup [src addr] [dst addr] [nb] [ttl]

[src addr]는 출발지의 IP주소이고, 0으로 설정하게 되면, 프로그램이 랜덤한 IP주소를 생성하여 소스 주소를 스푸핑하게 해준다. [dst addt]는 공격할 대상이 되는 시스템의 IP 주소이다. [nb]는 IDSwakeup이 내부적으로 실행시키는 iwu 도구에서 데이터그램을 공격 대상 시스템에 보낼 횟수이고, [ttl]은 time to live로서 패킷이 살아있는 시간을 의미한다.

3. 시뮬레이션 환경

3.1 환경구성

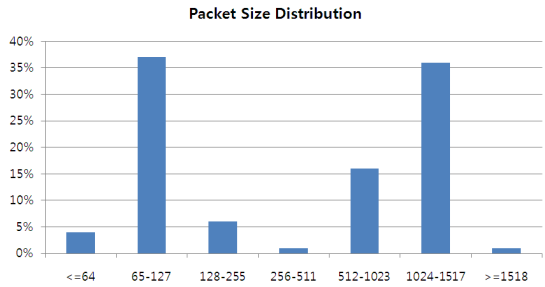
본 장에서는 점핑 윈도우 알고리즘을 이용하여 1100여개의 페이로드에 대하여 Snort 2.9.0 룰의 패턴을 매칭할 때 점핑 윈도우 크기에 따른 TCAM 엔트리 수와 평균 룩업 횟수를 측정하기 위해 (그림 5)와 같은 방법으로 시뮬레이션 환경을 구성하였다.



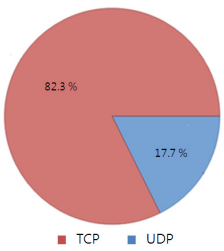
(그림 5) 시뮬레이션 환경

시뮬레이션 환경은 Snort 2.9.0에 존재하는 모든 룰을 파싱하여 패턴을 찾아 TCAM 테이블에 저장하는 부분과 인터넷 트래픽 패킷을 수집하는 부분으로 크게 나누어진다. 입력 패킷은 학교 망 내부 네트워크에 흐르는 인터넷 트래픽 중 페이로드가 존재하는 패킷과 IDSwakeup 공격 도구를 사용하여 발생된 패킷을 네트워크 분석 프로그램 와이어샤크[10]를 이용하여 생성한 1,100개의 패킷을 대상으로 한다. 그리고 Snort 2.9.0에 존재하는 모든 룰을 분석하여 이중 약 90%를 차지하는 'content' 옵션이 1~3개인 패턴 8,000여개를 수집하여 TCAM 엔트리 테이블을 생성하였다.

(그림 6)은 수집한 1,100개의 패킷 중 크기 별로 차지하는 분포율을 보여주고, (그림 7)은 TCP와 UDP 프로토콜 분포율을 나타낸다.



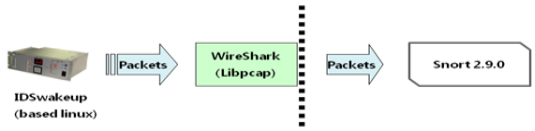
(그림 6) 패킷의 크기 분석(단위: 바이트)



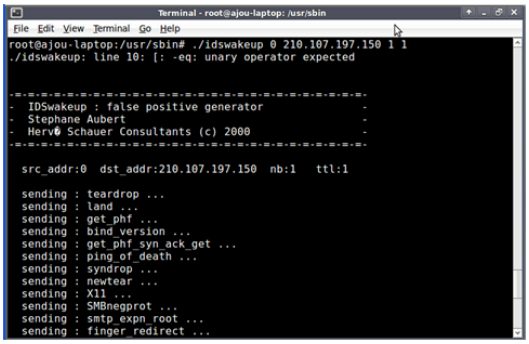
(그림 7) 패킷의 프로토콜 분석(단위 :%)

3.2 IDSwakeup을 이용한 페이로드 생성

IDSwakeup의 페이로드 생성 환경은 (그림 8)과 같이 두 대의 리눅스 시스템[12]에서 이루어 졌으며 공격 시스템인 IDSwakeup을 실행시켰을 때 (그림 9)과 같은 화면을 보여준다. IDSwakeup은 2.2절에서 언급한 여러 공격 기법을 이용하여 공격 대상 시스템인 Snort 2.9.0 IDS를 공격하고 와이어샤크를 이용하여 공격 트래픽을 수집한 패킷을 분석하여 페이로드를 저장한다. (표 2)는 IDSWakeup에서 공격할 시 생성되는 페이로드의 예제이다.



(그림 8) 패킷 수집 환경



(그림 9) IDSwakeup 실행 화면

(표 2) IDSwakeup 공격 시 페이로드 예제

/mylog.phtml
/cfide\\administrator\\startstop.html
/cfappman\\index.cfm
/admin_files\\order.log
/cgi-bin\\wrap
/cgi-bin/ph%66 HTTP/1.0
.....

4. 실험 및 결과 분석

4.1 실험방법

본 논문에서 점핑 윈도우 알고리즘 성능을 분석하기 위해 Snort 2.9.0 룰의 패턴을 이용하였다. 기존의 연구에서 적용한 Snort 2.1.0 룰의 패턴에 비해 전체 패턴의 수와 평균 패턴의 길이가 4배 이상 증가한 Snort 2.9.0에서 4~32-바이트 점핑 윈도우 알고리즘을 적용하였을 때 윈도우 사이즈에 따른 TCAM의 메모리 크기와 TCAM 룩업 횟수와의 상관관계를 분석한다.

4.2 Snort 2.9.0 룰 분석

평가를 위해 우선 Snort 2.9.0 룰로부터 모든 패턴을 파싱하여 분리하였다. 본 논문에서 제안된 구조는 고정 패턴만을 지원하기 때문에 'content' 옵션의 값만을 사용하여 패턴을 만든다. Snort 룰 파일에서 'root'와 같은 문자 값이나 '|20|'와 같은 바이트 값을 가지는 'content' 옵션의 패턴을 파싱하여 본 논문에서 사용할 패턴을 만들었다. (표 3)은 작성된 패턴의 예를 보여준다.

(표 3) 사용할 패턴의 예

.exe
<IMG
/in.php
public
|20|EMF
SpyDawn
root

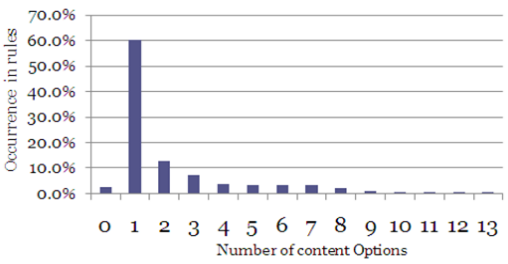
Snort에서 패턴 매칭 수행 부분은 전체 수행에서 70~80% 부분을 차지한다. 따라서 패턴의 수와 패턴의 길이는 패턴 매칭에 큰 영향을 주고 있다. (표 4)를 살펴보면 2006년에 배포된 Snort 2.3버전 이후로 최근 버전 까지 약 7배의 패턴이 급격하게 증가하고 있음을 알 수 있다.

(표 4) Snort 룰의 증가 경향

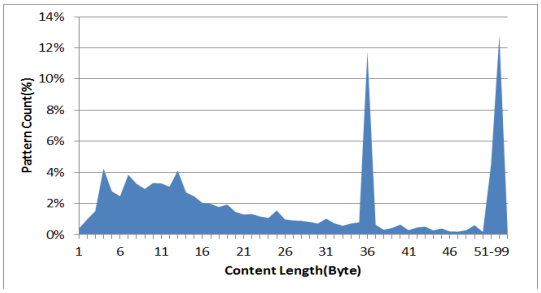
Rulesets	Total	Static Pattern
Snort 2.9(Nov, 2010)	16,016	9,937
Snort 2.8(June, 2008)	12,841	8,800
Snort 2.4(Jan, 2007)	5,047	3,432
Snort 2.3(Mar, 2006)	2,489	2,188
Snort 2.1(Dec, 2004)	1,046	942
Snort 1.9(Dec, 2003)	974	909

(그림 10)은 2010년에 배포된 Snort 2.9.0의 9,937개의 룰에서 각각의 룰에 존재하는'content'옵션에 대한 분포도를 보여준다. 전체 룰 중 'content'옵션을 가지지 않는 룰이 약 2.5% 이며, 한 개 이상의 'content'옵션을 포함한 룰이 95%로 이상을 차지하고, 최대 13개의 'content' 옵션을 갖는 룰이 존재한다. 이는 다수의 'content' 옵션을 가질 수 있으며 패턴 매칭 시 요구하는 'content'의 수만큼 전부 매칭이 이루어져야 패턴 매칭이 될 수 있다. 'content' 옵션을 가지는 룰 중 하나의 'content' 옵션을 가지는 비율은 62.7% 이고 1~3개의 'content' 옵션을 가지는 비율은 83.8%를 차지한다.

(그림 11)은 Snort 2.9.0 룰의 패턴 길이별 분포도를 나타내며, 16바이트 이하의 패턴이 차지하는 비율은 43.46%이고, 32바이트 이하의 패턴이 차지하는 비율은 63.06%이다. 100바이트 이상의 긴 패턴이 갖는 비율은 17.7%로이며 9,937개의 Snort 룰에서 패턴들의 중복을 제거하면 TCAM 메모리에 저장 할 8,107개의 패턴이 되고 패턴의 평균 바이트는 42바이트이다.



(그림 10) Snort 2.9.0의 'content' 옵션 분포도



(그림 11) Snort 2.9.0 룰의 패턴(content) 길이별 분포도

4.3 시뮬레이션 결과 및 비교분석

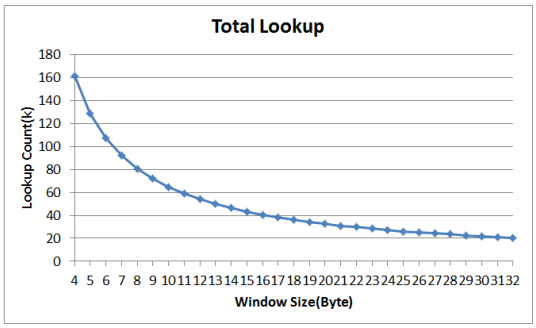
본 절에서는 Snort 2.9.0의 8100여개의 패턴을 TCAM에 저장할 경우 윈도우 크기(M)에 따른 TCAM의 크기를 분석하고 기존 연구와 비교 분석한다. 또한 시뮬레이션 환경을 이용하여 수집한 1,100개의 패킷 페이로드를 대상으로 점핑 윈도우 알고리즘을 적용하여 패턴 매칭을 수행했을 때 윈도우 크기에 따른 평균 TCAM 룩업 횟수를 분석하고 기존 연구와 비교해 본다.

(표 5)와 (그림 12)는 M-바이트 윈도우 크기 별 평균 TCAM 룩업 횟수를 보여준다. 본 시뮬레이션 결과 M의 크기가 커질수록 평균 TCAM 룩업 횟수가 급격하게 줄어들어 성능이 증가함을 볼 수 있다.

(표 6)은 검출한 1,100개의 패킷의 평균 페이로드 길이(634바이트)에 대한 윈도우 크기별로 룩업 횟수(평균 페이로드 길이/M)를 계산한 데이터를 보여준다. (표 5)와 (표 6)의 결과 값을 비교해 볼 때 (표 5)의 시뮬레이션 결과값이 평균적인 계산값 보다 약 1~3번의 룩업이 추가적으로 수행됨을 알 수 있다. 추가적인 룩업의 원인은 (표 6)의 계산값은 패턴이 바로 일치하거나 일치하지 않는 경우를 가정하고 단순 나눗셈한 값이지만 실제로는 하나의 패턴이 서브 패턴으로 나누어져 매칭이 될 수 있기 때문이다.

(표 5) Snort 2.9.0에서 윈도우 크기별 평균 룩업 횟수

M	Lookup Count	M	Lookup Count	M	Lookup Count	M	Lookup Count
4	161.16	12	54.05	20	32.64	28	23.57
5	128.85	13	49.79	21	31.03	29	22.68
6	107.58	14	46.47	22	29.71	30	21.95
7	92.3	15	43.28	23	28.47	31	21.26
8	80.77	16	40.58	24	27.28	32	20.5
9	71.9	17	38.27	25	26.16		
10	64.77	18	36.16	26	25.14		
11	58.96	19	34.25	27	24.3		



(그림 12) Snort 2.9.0에서 윈도우 크기별 평균 룩업 횟수

즉, 페이로드에서 패턴을 검출할 때 패턴이 부분적으로 매칭된 이후 연속적으로 이어지는 서브 패턴이 존재하는지 찾기 위해서 서브 패턴에 대한 룩업을 추가적으로 수행해야 한다. 전체 페이로드에 패턴의 서브패턴이 차지하는 빈도가 높거나 패턴의 길이가 길어질수록 패턴 매칭 룩업 횟수가 많아지게 된다.

(표 6) 평균 페이로드 크기(634바이트)에 대한 윈도우 크기별 룩업 횟수 (634/M)

M	Lookup Count	M	Lookup Count	M	Lookup Count	M	Lookup Count
4	158.50	12	52.83	20	31.70	28	22.64
5	126.80	13	48.77	21	30.19	29	21.86
6	105.67	14	45.29	22	28.82	30	21.13
7	90.57	15	42.27	23	27.57	31	20.45
8	79.25	16	39.63	24	26.42	32	19.81
9	70.44	17	37.29	25	25.36		
10	63.40	18	35.22	26	24.38		
11	57.64	19	33.37	27	23.48		

(표 7)은 기존 연구에서 Snort 2.1.0을 대상으로 시뮬레이션 한 평균 룩업 횟수이다. 평균 룩업 횟수가 본 논문의 값보다 큰 이유는 페이로드의 크기가 평균 690정도로 본 논문에서 사용한 페이로드보다 크기 때문이다. 기존 연구에서 도출한 (표 7)의 TCAM 룩업 횟수와 본 논문에서 시뮬레이션한 (표 5)의 데이터를 분석해보면 모두 윈도우 사이즈4~10까지의 룩업 횟수가 급격히 감소하고 있으며, 16~32는 룩업 횟수가 조금씩 감소하고 있는 모습을 볼 수 있다. 이는 페이로드에 길이가 긴 패턴이 나타나는 빈도수가 많지 않기 때문으로 분석된다.

(표 7) Snort 2.1.0에서의 윈도우 크기별 평균 룩업 횟수

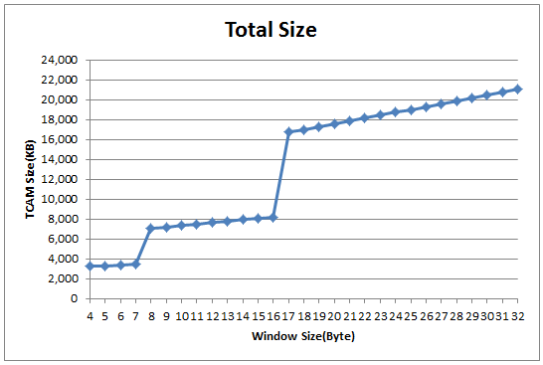
M	Lookup Count	M	Lookup Count	M	Lookup Count	M	Lookup Count
4	193	12	65	20	39	28	29
5	154	13	60	21	37	29	28
6	129	14	55	22	36	30	26
7	110	15	52	23	34	31	26
8	96	16	48	24	32	32	24
9	86	17	46	25	32		
10	77	18	44	26	31		
11	70	19	41	27	30		

윈도우 크기가 커지면 평균 룩업 횟수 감소하지만 점핑 윈도우 알고리즘에서는 하나의 패턴이 가질 수 있는 모든 조합의 시프트된 패턴이 필요하기 때문에 M의 크기가 커질수록 시프트된 패턴의 수가 늘어나 TCAM의 엔트리 수는 증가함을 (표 8)과 (그림 13)에서 볼 수 있다.

(표 8) Snort 2.9.0 에서의 윈도우 크기 별 요구되는 TCAM 크기 (단위: KB)

M	TCAM Size	M	TCAM Size	M	TCAM Size	M	TCAM Size
4	3,251	12	7,647	20	17,584	28	19,873
5	3,323	13	7,790	21	17,870	29	20,159
6	3,395	14	7,933	22	18,155	30	20,,446
7	3,466	15	8,076	23	18,441	31	20,731
8	7,074	16	8,219	24	18,728	32	21,017
9	7,218	17	16,725	25	19,014		
10	7,360	18	17,012	26	19,300		
11	7,504	19	17,297	27	19,585		

(표 8)에서 M이 8과 17일 경우 TCAM 크기가 급격하게 증가하는 것은 TCAM의 룩업 단위가 36, 72, 144, 288 비트로 제한되어 있다는데 기인한다. 이중 16비트를 상태정보 (2.1절에서 설명한 바와 같이 다음 서브패턴을 찾을 때 사용하는)를 저장하는 용도로 사용하면 실제로는 M이 4~7일 때 72 비트, 8~16일 때 144 비트, 17~34일 때 288 비트형 TCAM을 사용해야 한다. 따라서 M이 7인 경우와 8인 경우 실제 TCAM 엔트리는 크게 차이가 없으나 TCAM의 룩업 단위가 2배가량 증가하기 때문에 TCAM 메모리도 2배가량 늘어나게 된 것이다.



(그림 13) Snort 2.9.0에서의 윈도우 크기별 요구되는 TCAM 크기

(표 9) Snort 2.1.0 에서의 윈도우 크기 별 요구되는 TCAM 크기 (단위: KB)

M	TCAM Size	M	TCAM Size	M	TCAM Size	M	TCAM Size
4	122	12	424	20	1,173	28	1,490
5	133	13	445	21	1,212	29	1,529
6	146	14	466	22	1,251	30	1,568
7	157	15	486	23	1,292	31	1,607
8	335	16	507	24	1,338	32	1,647
9	356	17	1,053	25	1,372		
10	377	18	1,092	26	1,411		
11	401	19	1,132	27	1,451		

(표 9)는 Snort 2.1.0에 대한 윈도우 크기별 요구되는 TCAM의 크기를 분석한 것이다. Snort 2.1.0의 패턴의 수는 2,247개 이고, 4바이트 이하의 패턴이 갖는 분포율은 약 50%, 16바이트 이하의 패턴이 갖는 분포율은 약 90% 이상이며 전체 패턴의 평균 패턴의 길이는 약 9바이트이다. 4장 2절에서 소개한 바와 같이 Snort 2.9.0는 Snort 2.1.0에 비해 패턴의 수와 패턴의 평균 크기에 있어서 4배 정도 크다. (표 8)과 (표 9)에 따르면 Snort 2.9.0 환경

에서 요구되는 TCAM의 크기는 대략적으로 13~26배 정도를 더 요구한다. 이것은 패턴의 수가 증가하고 또 각 패턴의 길이가 증가함에 따라 매치해야 할 서브 패턴이 크게 증가하기 때문이다.

Snort 2.1.0에서는 패턴의 수와 룩업 횟수를 고려하여 9Mb TCAM에서 16바이트 점핑 윈도우를 사용하면 TCAM 엔트리를 모두 수용 가능한 최적의 성능을 낼 수 있었다. Snort 2.9.0에서 패턴의 개수와 패턴의 평균 길이가 증가함에 따라 요구되는 TCAM 엔트리가 크게 증가하여 9Mb TCAM에서 16-바이트 점핑 윈도우를 적용하기에는 부적절하다. (표 8)에서 보면 M=16일 경우, 대략 63Mb 정도의 TCAM을 요구한다. TCAM은 일반적으로 1Mb, 2Mb, 4.5Mb, 9Mb, 18Mb 등의 크기를 지원하는 것을 고려하면 대략 18Mb TCAM 4개가 필요로 한다. 제품에 따라 차이가 있지만 TCAM은 8개 까지 캐스케이딩 방법으로 연결할 수 있다. 따라서 Snort 2.9.0의 경우 16-바이트 윈도우 크기에서 4개의 18Mb TCAM을 연결하면 모든 패턴 엔트리를 수용하면서 최적의 성능을 낼 수 있다는 것을 알 수 있다.

5. 결 론

본 논문에서는 인터넷의 확장과 빠른 네트워크 속도로 인해 증가하는 분산 서비스 공격, 인터넷 웜, 이메일 바이러스 등의 악의적인 공격을 효과적으로 탐지하기 위해 대표적인 침입 탐지 시스템인 Snort의 룰의 패턴도 늘어나고 패턴의 길이도 길어지고 있다. 본 논문에서는 Snort 2.9.0 룰의 패턴들을 분석하고 기존의 TCAM 기반 점핑 윈도우 패턴 매칭 알고리즘을 활용하여 Snort 2.9.0의 패턴 매칭의 메모리 효율성 및 성능을 분석하였다.

기존 Snonrt 2.10 환경에서의 패턴 매칭 알고리즘의 성능과 비교하기 위해 시뮬레이션 환경을 구축하였다. 학내 망의 인터넷 트랙픽과 IDSwakeup 도구를 이용하여 약 1,100여 개의 패킷을 수집하고 페이로드를 생성한 후, 다양한 크기의 윈도우 사이즈에 대하여 시뮬레이션 한 결과를 바탕으로 윈도우 크기에 따른 TCAM 크기와 전체 페이로드의 TCAM 룩업 횟수를 측정하여 각각의 윈도우 크기별 평균 룩업 횟수에 대해 알아보았다.

패턴의 길이가 늘어나고 패턴 수가 증가할수록 점핑 윈도우 알고리즘을 적용하였을 때 매칭되는 서브 패턴의 빈도수가 높아지게 되고 그에 따른 TCAM 룩업 횟수가 증가하게 되며, 윈도우 사이즈가 커지면 커질수록 룩업 횟수는 줄어들지만 TCAM 엔트리에 저장되는 서브 패턴이 늘어나는 동시에 엔트리 수가 증가하여 TCAM의 메모리를 더 많이 요구 하게 된다. 본 논문의 실험결과에 따르면 Snort 2.9.0의 경우 16-바이트 윈도우 크기에서 4개의 18Mb TCAM을 연결하여 사용하면 모든 패턴 엔트리를 수용하면서 최적의 성능을 낼 수 있다는 결과를 도출하였다.

앞으로 침입탐지 시스템에서 탐색해야 하는 패턴의 수는 지속적으로 증가할 것이다. 이 경우 기존의 TCAM 기반의 매칭 알고리즘으로는 성능 향상에 한계에 도달하 것으로 예상된다. 이 문제를 해결하기 위해서는 매칭의 병렬성을 보다 더 높일 수 있는 방법을 연구해야 할 것이다.

참 고 문 헌

[1] SNORT network intrusion detection system, http://www.snort.org

[2] Martin Roesch, Chris Green, Sourcefire, Inc, “SNORT Users Manual”, 2.9.0 The Snort Project December 2, 2010

[3] 해킹/바이러스 통계 , ″2010년 10월 인터넷침해사고 동향 및 분석 월보 ″, Korea Internet & Security Agency 2010년

[4] Fang Yu, ″High Speed Deep Packet Inspection with Hardware Support″ Computer Science in the GRADUATE DIVISION of the UNIVERSITY OF CALIFORNIA, BERKELEY 2006

[5] F. Yu, R. H. Katz and T. V. Lakshman, “Gigabit Rate Packet Pattern-Matching Using TCAM,” Proceedings of the 12th IEEE International Conference on Network Protocols, pp.174-183, 2004

[6] M.Gao, K. Zhang and J.Lu, ″Efficient Packet Matching for Gigabit Network Intrusion Detection using TCAMS,″ Proceedings of the 20th International Conference on Advanced Information Networking and Applications -Volume1, 2006

[7] 성정식, 강석민, 이영석, 권택근, 김봉태, “TCAM을 이용한 고성능 패턴 매치 알고리즘” 정보처리학회논문지© 제12-C권, 제4호, pp.503-510, 2005년 8월

[8] False attack generator IDSwakeup, http://www.hsc.fr

[9] Security Onion LiveCD, Doug Burks, http://securityonion.blogspot.com/

[10] NetworkAnalysisTools http://www.wireshark.com international Conference on advanced information system engineering, January 2006.
[11] 한국정보보호진흥원, "Snort를 이용한 IDS구축", 2005
[12] Ubuntu Server Guide, "http://www.ubuntu.com/", Canonical Ltd. and members of the Ubuntu Documentation Project3 2008

◐ 저 자 소 개 ◑

류 기 열
1985년 서울대학교 전자계산기공학과 졸업(학사)
1987년 한국과학기술원 전산학과 졸업(석사)
1992년 한국과학기술원 전산학과 졸업(박사)
1994년~현재 아주대학교 정보컴퓨터공학부 교수
관심분야 : 서비스지향컴퓨팅, 유비쿼터스시스템, 프로그래밍언어, 시스템보안 etc.
E-mail : kryu@ajou.ac.kr.

이 성 윤
2007년 전남과학대학 졸업(학사)
2011년 아주대학교 대학원 지식정보보안학과 졸업(석사)
2011년~현재 퓨쳐시스템 정보통신연구소 전임연구원
관심분야 : Web DDoS, Snort, 모바일 보안, etc.
E-mail : hwauni@ajou.ac.kr.