The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 2, pp.29-37, Apr. 30, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.2.29
JIIBC 2020-2-4

머신러닝 기반의 자동 정책 생성 방화벽 시스템 개발

Development of Firewall System for Automated Policy Rule Generation based on Machine learning

한경현*, 황성운**

Kyung-Hyun Han*, Seong-Oun Hwang**

요 약 기존에 사용되던 방화벽들은 기본적으로 정책을 수동적으로 입력해 주는 방식으로 되어 있어 공격이 오는 즉시 대응하기 쉽지 않다. 왜냐하면 전문 보안 관리자가 이를 분석하고 해당 공격에 대한 방어 정책을 입력해 주어야하기 때문이다. 또한, 기존 방화벽 정책은 공격을 막기 위해 정상 접속까지 차단하는 경우가 많다. 패킷 자체는 정상적이지만 유입량이 많아 서비스 거부를 발생시키는 공격이 많기 때문이다. 본 논문에서는 방어 정책을 입력하는 부분을 인공지능으로 대체하여 정책을 자동으로 생성하고, 정상 접속 학습을 통해 생성된 화이트리스트 정책으로 정상 접속은 가능하면서 Flooding, Spoofing, Scanning과 같은 공격만을 차단하는 방법을 제안한다.

Abstract Conventional firewalls cannot cope with attacks immediately. It is because security professionals or administrators need to analyze them and enter relevant policies to the firewalls. In addition, those policies may often block even normal accesses. Even though the packet themselves are normal, there exist many attacks that cause denial of service due to the inflow of a large amount of those packets. In this paper, we propose a method to block attacks such as Flooding, Spoofing and Scanning while allowing normal accesses based on whitelist policies which are automatedly generated by learning normal access patterns.

Key Words : Firewall, Intrusion Detection, Machine Learning

Ⅰ. 서 론

현재 우리 사회는 다양한 전자제품이 산업 현장 및 실생활 환경에서 사용되고 있다. 특히 최근에는 IoT 기술이 발전함에 따라 인터넷 네트워크와 연결되는 스마트 TV, 스마트 냉장고, 스마트 자동차 등 수많은 제품들이 출시되어 우리 생활을 보다 편하게 해주고 있다.

그러나 보안을 고려하지 않으면 많은 피해를 입히게 된다. 실제로 2017년 6월에도 은행 공동망을 운영하는 금융결제원에 분산 서비스 거부 공격(DDoS) 공격이 발생했다[1]. 해당 사건은 DDoS라는 대표적인 네트워크 공격 방식을 통해 발생했다. 이를 통해 각 기관의 서비스가 마비되지는 않았으나 모든 금융 거래가 중지되어 큰 혼란이 발생할 수 있는 위험한 상황이었다. 포털 사이트의 경우에도 피해가 있겠지만, 특히 막대한 돈이 오가는 은행 네트워크로의 공격은 치명적이다.

*준회원, 홍익대학교 전자전산공학과
**정회원, 가천대학교 컴퓨터공학과
접수일자 2019년 10월 29일, 수정완료 2020년 2월 23일
게재확정일자 2020년 4월 3일

Received: 29 October, 2019 / Revised: 23 February, 2020 / Accepted: 3 April, 2020
*Corresponding Author: sohwang@gachon.ac.kr
Dept. of Computer Engineering, Gachon University, Korea

위의 사례는 우리나라의 다양한 산업과 실행활의 단편적인 예일 뿐이다. 위 사례와 같은 큰 기관이나 사이트의 경우에는 전문 보안 팀을 운영하지만, 가정에서 혹은 개인적으로 IoT장비나 네트워킹이 가능한 장비를 사용하는 경우에는 피해를 알아차리지 못할 수도 있다. 하지만 기존의 방화벽은 전문가의 개입이 없이는 제 기능을 하기에 무리가 있다. 규모가 있는 회사에서는 방화벽 관리를 위해 전문 인력을 사용하지만, 개인 사용자에게는 전문적인 지식이 떨어짐으로 인해 방화벽이 제 기능을 하지 못하는 경우가 많다. 또한 방화벽이 정책을 직접 입력해주어야 하는 구동 원리상 해킹의 위험보다 뒤늦게 반응하게 된다는 한계점이 존재한다. 이러한 문제를 해결하기 위해 자동으로 방화벽 정책을 추가하는 연구가 진행되었다.

II. 관련 연구

2004년 Ertoz 등은 네트워크 공격을 탐지하기 위해 Minnesota Intrusion Detection System (MINDS)를 제안했다[2]. MINDS는 먼저 레이블(label)된 데이터 세트와 클러스터링을 통해 비정상적인 공격을 탐지한다. 또한 해당 공격 트래픽을 연관 규칙으로 마이닝하여 탐지 정책을 생성한다. 이 연구는 침입 탐지 모듈에 새로운 탐지 정책을 생성하는데 연관 규칙 마이닝이 유용하다는 것을 보여준다. 네트워크 분석가는 생성된 정책 중에서 성능이 우수한 정책을 선택할 수 있다.

2010년 이영석은 DDoS 공격에 대응하기 위한 프레임 워크를 설계하고 구현하였다[3]. 이 연구에서는 액티브 네트워크를 기반으로 위조 IP 공격, DDoS 공격에 대응할 수 있는 프레임워크를 제안하였다. 하지만 프레임워크는 각 기업마다 설치하기에 어려운 점이 있다.

2015년 Nattawat Khamphakdee 등은 연관 규칙 마이닝을 사용하여 Snort에서 사용되는 탐지 정책을 생성하였다[4]. 이 연구에서는 DARPA 1999 데이터 세트를 사용했다. 이 연구에서 생성되는 탐지 정책의 정확성은 데이터 집합의 속성 수에 비례하여 증가하는 것으로 확인되었다. 이로써, 생성된 규칙의 정확성을 높이기 위해 데이터 세트의 속성의 수를 늘려야 한다고 설명한다.

2015년 심규석 등은 네트워크 트래픽을 구분하는 Snort 규칙을 자동으로 생성하기 위해 시퀀스 마이닝을 적용했다[5]. 이 연구에서는 수집된 패킷의 페이로드를 시퀀스 마이닝으로 분석하여 각 응용 프로그램의 패킷을 구분할 수 있는 문자열을 찾고, 이를 조합하여 Snort 규칙으로 자동 생성하였다. 그 결과 평균 98.14%의 정확도로 응용 프로그램을 구별할 수 있었다. 하지만, 이 기술로 악성 프로그램에서 생성된 패킷을 구분하는 실험은 아직까지 없는 실정이다.

2018년 김현석은 산업 제어 시스템 보안을 위한 비정상행위 탐지 시스템을 구현하였다[6]. 기술이 발전됨에 따라 폐쇄망을 이용하던 산업 제어 시스템도 네트워크가 개방되면서 이에 대한 공격에 대응할 수 있는 시스템이 필요하다. 하지만 이 연구에서도 탐지 정책을 이용하지만 이를 위한 자동 정책 생성은 되지 않는다.

2019년 최상용은 Suricata를 이용한 대용량 네트워크 트래픽 수집 시스템을 제안하였다[7]. 트래픽 수집 기술은 최신 환경에 맞춰 대용량 트래픽을 무손실로 수집할 수 있다. 하지만 트래픽 분석 및 정책 생성 기술의 발전이 이를 따라가지 못 하고 있다.

지금까지의 연구는 생성된 탐지 정책 중 적절한 정책을 선택하는 방법을 제시하지 않고, 정상 접속을 차단하지 않는 방법도 고려되지 않았다. 전문 지식이 없는 사용자가 쉽게 사용할 수 있고, 생성되는 방화벽 정책이 정상 사용자의 접속을 차단하지 않고 공격만을 차단하는 기술이 필요하다.

III. 인공지능 방화벽 시스템

1. 시나리오

대부분의 네트워크 공격은 정상적인 패킷과 큰 차이가 없이 IP 등 일부 정보만 다른 패킷을 대량으로 보내서 처리를 지연시키는 공격이다. 이런 이유로 네트워크 트래픽이 과도하게 발생하는 공격이 발생할 경우에는 공격 패킷과 정상 패킷 모두를 차단하는 정책이 들어가 있다. 만약 서버가 24시간동안 항상 중요한 작업을 해야 하는 곳에 배치되어 있다면 정상 패킷을 차단하지 않아야 한다. 본 연구를 통해 개발된 방화벽 시스템을 정상적인 패킷을 학습하여 화이트리스트 정책을 생성해 둠으로써 정상 패킷은 통과시키면서 공격을 차단하고자 한다.

2. 개발 요구사항

연구목적에 맞게 방화벽을 개발하기 위해 아래와 같은 요구사항을 도출하여 설정하였다.

* 자동으로 정책을 생성, 적용할 수 있어야 한다.

* 네트워크 공격이 발생할 경우 공격성 패킷은 차단하고 정상 패킷만 허용할 수 있어야 한다.
* 정책의 개수를 최소화하여 방화벽이 실행되는데 과부하가 없도록 해야 한다.

3. 기초 설계

본 연구의 시스템 구성도는 Fig 1과 같다. Filtering을 위한 INPUT, FORWARD, OUTPUT chain의 DROP 정책을 통과하여 최종적으로 POSTROUTING chain에 도달한 packet을 정상 packet으로 본다. log 파일을 생성한 후에 Apriori 알고리즘[8]을 실행할 수 있는 형태(.csv)로 변환한다. 그리고 Apriori 알고리즘을 실행시킨 후에 나오는 rule을 기반으로 Iptables 정책을 생성한다. 그 후에 PREROUTING chain에 적용시키는 작업으로 모든 과정이 자동화로 진행된다.

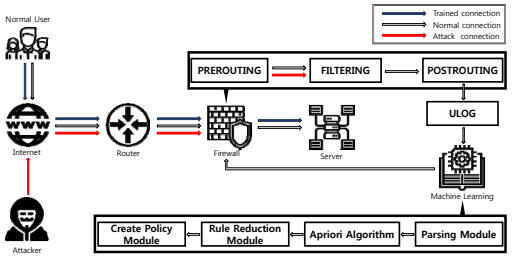

그림 1. 시스템 구조
Fig. 1. System structure

시스템 구성도에 대한 설명은 아래와 같다.

* ULOG

ULOG는 패킷에 대한 log를 특정 디렉터리에 저장하기 위한 용도로 사용한다. 차단되지 않은 패킷은 정상으로 볼 수 있다.

* Parsing Module

ULOG의 출력은 protocol마다 필드 수가 다르기 때문에 log를 Parsing해서 protocol별로 패킷을 분류하여 csv 형태의 파일로 변환한다.

* Apriori 알고리즘

생성된 csv 파일로 Apriori 알고리즘을 실행하여 정상 패킷에 대한 rule을 생성한다.

* Rule Reduction Module

Apriori 알고리즘을 통해 나온 Rule을 일정한 기준에 따라 선별한다.

* Create Policy Module

최종적으로 선별된 Rule을 방화벽 정책에 알맞은 형태로 변환 후 정책에 적용한다.

가. Parsing Module

네트워크 공격을 차단하는 Chain을 통과하고 POSTROUTING Chain에 도착한 패킷들은 공격이 아닌 정상 패킷으로 판단하고 화이트리스트(whitelist) 기반의 정책을 생성하기 위해 ULOG를 이용하여 LOG 기록을 남긴다.

기본적으로 LOG 기록이 남게 되는 파일이 아닌 임의의 지정한 경로, 파일에 LOG 기록을 남기기 위해 ULOG를 사용하며, 이를 사용하기 위한 옵션은 아래 Table 1과 같다.

표 1. ULOG 옵션
Table 1. ULOG option

Option	Meaning
--nflog-group	Set a group number from 1 to 32 to each IPtable policy for storing log to specific path.
--nflog-prefix	Set a message and output the message to the front of each log item.

ULOG를 이용해 만든 LOG 파일의 확장자로는 Apriori 알고리즘을 동작시킬 수 없다. 따라서 Apriori 알고리즘 동작을 위해 CSV 확장자로 변경하는 작업을 거쳐야 한다. 이때 문자열 Parsing을 이용해 불필요한 항목을 제거하고 알고리즘 동작 시 Class로 사용할 항목을 추가한다.

나. Apriori 알고리즘

Apriori 알고리즘은 기본적으로 CSV 형태의 데이터를 다룬다. 그러므로 방화벽의 log 수집 단계에서 얻은 CSV 형태의 데이터를 기준으로 알고리즘을 실행, Rule을 생성한다.

본 연구에서 사용하려는 Apriori 알고리즘은 Borgelt 등이 작성한 Apriori 알고리즘[9]으로, 이 프로그램은 C 언어로 작성되었다. 이외에도 Python이나 Ruby등 다양한 언어로 작성된 Apriori 알고리즘이 존재하지만 C언어로 구현된 알고리듬이 추후 수정이 용이하기 때문에 이를 사용한다.

Rule 생성 자체는 알고리즘의 연산에 의해서 생성되지만 해당 알고리즘에는 다양한 옵션이 존재한다. 이 중 필요한 옵션만을 사용하는데 해당 옵션은 아래 Table 2와 같다.

표 2. Apriori 옵션
Table 2. Apriori option

Option	Meaning	Default
-t#	target type : frequent item sets : closed (frequent) item sets : maximal (frequent) item sets : (frequent) generators : association rules	s
-s#	minimum support of an item set positive: percentage of transactions negative: absolute number of transactions	10
-c#	minimum confidence of a rule as a percentage	80
-R#	read item selection/appearances from a file parameter: file name	
-k#	item separator for output	" "

Table 2의 옵션에 대한 설명은 아래와 같다.

-t#: 얻고자 하는 결과를 선택하는 옵션이다. 본래 Borgelt의 Apriori 알고리즘은 하나의 결과만 만들어내는 알고리즘이 아니다. 총 5개의 결과를 얻을 수 있는데 s(빈번한 아이템 세트), c(닫힌 빈번 항목 세트), m(최대 빈번 항목 세트), g(빈번 발생기), r(연관규칙)이 있다.

이 중 방화벽 Log에서 얻은 정보를 통해 공격이 아닌 패킷들에 대한 아이템 세트들의 연관 규칙을 찾아야 하므로 이 프로젝트에서는 r 옵션을 사용한다.

-s#: 최소 Support를 지정한다. 양수의 경우 트랜잭션의 비율, 음수의 경우 트랜잭션의 절대값을 기준으로 한다.

-c#: 최소 Confidence를 지정한다.

-R#: R 옵션은 출력의 형식을 정하는 옵션이다. 출력의 형식을 정해 놓은 텍스트 파일을 읽어 들여 Rule 생성에 반영한다.

-k#: 출력을 위한 항목 구분 기호이다.

위의 옵션들을 사용해서 Apriori 알고리즘을 사용하려면 ubuntu 기준으로 아래와 같은 명령어를 사용하면 된다. 이 명령어는 연관규칙이면서 최소 Support 30%, 최소 Confidence 20%인 Rule들을 반점(,) 구분기호로 구분하며 R옵션에 명시된 형태로 Rule을 출력한다.

./apriori -tr(연관규칙) -s30 -c20 -k, [데이터 csv 파일 이름] [출력 파일 이름] -R [R옵션을 위한 텍스트파일]

다. Rule Reduction Module

생성되는 Rule은 빈번한 아이템 집합을 모두 출력하기 때문에 아이템의 수가 많아질수록 Rule의 출력 개수 또한 기하급수적으로 증가한다. 출력되는 모든 Rule을 방화벽 정책에 추가하는 것은 시스템에 과부하를 발생시키고, 연산능력 자체를 떨어뜨릴 수 있는 위험이 있다. 그러므로 생성되는 Rule 중 비교적 넓은 범위의 패킷을 다룰 수 있는 몇개의 Rule을 정책에 적용할 필요성이 있다.

Rule의 축소를 위해서는 몇 가지 기준이 되는 규칙이 필요하다.

1) Confidence가 100인 Rule만 남긴다.

그 이유는 정상적인 패킷에 대한 패턴을 찾는 방식을 사용하므로 Confidence가 100이하인 경우 정상적인 패킷까지 방화벽에서 차단하는 오탐이 발생할 수 있기 때문이다.

2) 보다 넓은 범위의 Rule을 남긴다.

Apriori 알고리즘을 사용 시 모든 아이템의 조합이 출력되는데 출력되는 방식이 보다 위쪽에 자리한 Rule일수록 Support 범위가 좁고, Confidence가 높고 아이템이 많다. 그리고 이 Rule의 아래에 자리한 Rule일수록 위쪽에 자리엔 Rule의 탐지 범위를 포함하고, Confidence가 보다 낮아지며 아이템 수가 적어지게 된다. 그렇기 때문에 보다 넓은 범위를 탐지할 수 있는 Rule을 사용함으로 써 적용할 정책의 개수를 줄일 수 있다. Confidence가 낮아지는 문제점이 존재하지만 이는 처음에 Confidence가 100인 Rule만 남김으로써 해결할 수 있다.

라. Create Policy Module

Apriori 알고리즘 동작 후 생성된 Rules는 IPtables에 적용할 수 없다. 따라서 생성된 Rules를 IPtables 형식에 맞게 수정하는 작업이 필요하다. 정책 생성에 필요한 적절한 항목을 선택하여 화이트리스트 기반의 ACCEPT 룰을 생성하고 IPtables 정책을 저장하는 형식에 맞게 작성하고 중복된 정책들을 제거한다.

4. 상세 설계 및 구현

가. 구현 환경

구현을 위해 필요한 환경은 아래의 Table 3과 같다.

표 3. 구현 환경
Table 3. Implementation environment

Virtual Machine	VMware WorkStation 12		
OS	Ubuntu 13.10	Kernel Version	v1.4.18
		IPtables Version	Linux 3.11.0-12-generic
Programming Language	C, Bash		

나. Parsing Module 구현

생성된 log파일을 Apriori 알고리즘을 실행시키기에 알맞은 형태인 csv 확장자로 변환해주는 작업이 필요하며, 이 과정에서 각각의 protocol마다 가지고 있는 field의 수가 다르다. 이로 인해 Apriori 알고리즘을 실행하는 과정에서 오류가 발생할 수 있기 때문에 각각의 packet log에 대해 protocol을 기준으로 분류하여 csv파일을 생성해줄 필요가 있다. 아래의 소스코드를 실행하면 그 결과 값으로 ICMP.csv, TCP.csv, UDP.csv 등 protocol 별로 알고리즘을 동작시킬 수 있는 CSV파일이 생성된다.

아래의 Fig 2 코드는 각각의 TCP.csv, UDP.csv, ICMP.csv에 우선적으로 field명을 입력해주기 위한 과정이다.

```
        FILE *file;
        FILE *tcp;
        FILE *udp;
        FILE *icmp;
        int i, numOfLine = 0;
        char String[MAX];
        tcp = fopen("../CSV/TCP.csv", "a");
        fprintf(tcp, "Attack , IN , OUT , MAC , SRC , DST , LEN ,
TTL , PROTO , SPT , DPT\n");
        fclose(tcp);
        udp = fopen("../CSV/UDP.csv", "a");
        fprintf(udp, "Attack , IN , OUT , MAC , SRC , DST , LEN ,
TTL , PROTO , SPT , DPT\n");
        fclose(udp);
        icmp = fopen("../CSV/ICMP.csv", "a");
        fprintf(icmp, "Attack , IN , OUT , MAC , SRC , DST , LEN
, TTL , PROTO\n");
        fclose(icmp);
```

그림 2. Field 이름 설정
Fig. 2. Field name setting

아래의 Fig 3 소스코드는 문자열 파싱후 각각의 protocol에 대해 Attack=NO라는 라벨링을 하기 위한 과정이다

```
                if (TCP[0] != 0){
                                fprintf(tcp,"Attack=NO , %s , %s
, MAC=%s , %s , %s , %s , %s , %s , %s , %s\n", IN, OUT, MAC, SRC
, DST, LEN, TTL, TCP, SPT, DPT);
                                }
                else if (UDP[0] != 0){
                                fprintf(udp,"Attack=NO , %s , %s
, MAC=%s , %s , %s , %s , %s , %s , %s , %s\n", IN, OUT, MAC, SRC
, DST, LEN, TTL, UDP, SPT, DPT);
                                }
                else if (ICMP[0] != 0){
                                fprintf(icmp,"Attack=NO , %s , %s
, MAC=%s , %s , %s , %s , %s , %s\n", IN, OUT, MAC, SRC, DST, LE
N, TTL, ICMP);
                                }
                pt1 = strtok(NULL, deli);
```

그림 3. 파싱과 라벨링
Fig. 3. Parsing and labeling

아래의 Fig 4 소스코드는 파싱 후에 생성된 TCP.csv, UDP.csv, ICMP.csv를 Apriori 알고리즘으로 실행시켜 (protocol).out 이라는 txt형식으로 다시 저장시키는 과정이다.

```
        system("./apriori -tr -k, ../CSV/TCP.csv ../CSV/TCP.out -
R ../shell/Roption_attno.txt");
        system("./apriori -tr -k, ../CSV/UDP.csv ../CSV/UDP.out -
R ../shell/Roption_attno.txt");
        system("./apriori -tr -k, ../CSV/ICMP.csv ../CSV/ICMP.out
-R ../shell/Roption_attno.txt");
```

그림 4. Apriori 실행
Fig. 4. Apriori execution

다. Apriori 알고리즘 구현

Apriori 알고리즘에서 정상 패킷을 학습하는 Rule 생성을 위해 –R 옵션을 설정해주어야 한다. Fig 5는 -R 옵션에 사용하는 Apriori Rule 생성 결과의 형태 결정 파일이다. in 명령어를 통해 조건이 되는 아이템 세트는 모두 들어갈 수 있도록 설정 되어있고, Attack=NO out은 공격이 아닌 패킷에만 붙여지는 라벨을 기준으로 Attack=NO라는 패킷에 대한 아이템들의 연관 관계를 출력하라는 의미의 명령어이다.

```
root@ubuntu:/create_rule/shell# more Roption_attno.txt
in
Attack=NO out
```

그림 5. R 옵션 파일
Fig. 5. R option file

라. Rule Reduction Module 구현

Rule 축소는 자동적으로 Rule이 생성됨과 동시에 축소될 수 있도록 bash 쉘 스크립트 파일로 작성하였다. 이 중 이 보고서에는 축소에 필요한 핵심적인 부분만 작성한다. 이외에는 파일의 입출력 또는 해당 프로그램이 동작할 수 있도록 하는 부수적인 명령어들이 존재한다.

```
#Search for rules with an accuracy of 100 or less
j=0
for((i=0;i<${#ICMP_array[@]};i++))
do

        export DIFF=`echo "${ICMP_array[$i]}<100"|bc`
        if [ "$DIFF" -eq 1 ]
        then
                delete_ICMP_array[$j]=`expr $i + 1`
                j=`expr $j+1`
        fi
done
```

그림 6. confidence<100인 룰
Fig. 6. Rules with confidence<100

위 Fig 6은 Confidence가 100인 Rule만 추출하는 부분으로, Apriori 알고리즘을 통해 출력된 파일에서 Confidence 부분만 추출한 후 Confidence가 100이하인 Rule의 Index를 저장한다.

```
echo "[Delete a rule with an accuracy of 100 or less...]"
#Delete a rule with an accuracy of 100 or less
for((i=0;i<${#delete_ICMP_array[@]};i++))
do
        if [ ${#delete_ICMP_array[@]} -eq 0 ]
        then
                text_index=${delete_ICMP_array[$i]}
                command=(`sudo sed -i $text_index'd' ../CSV/ICMP.out`)
        else
                text_index=${delete_ICMP_array[$i]}
                text_index=`expr $text_index - $i`
                command=(`sudo sed -i $text_index'd' ../CSV/ICMP.out`)

        fi
done
```

그림 7. index를 통한 필터링
Fig. 7. Filtering by index

이후 Fig 7과 같이 index에 저장된 Rule의 index번호를 보고, Rule파일에서 직접 해당 index의 Rule을 삭제한다.

```
for((i=0;i<${#line_field_ICMP_array[@]};i++))
do
        k= expr "$i" + 1`
        if [ "$k" -eq "${#line_field_ICMP_array[@]}" ]
        then
                command=(`sudo sed -n $k'p' ../CSV/ICMP.out >> ../CSV/result_ICMP.out`)
        else
                export Compare=`echo "${line_field_ICMP_array[$i]}<${line_field_ICMP_array[$k]}"|bc`
                if [ "$Compare" -eq "1" ]
                then
                        command=(`sudo sed -n $k'p' ../CSV/ICMP.out >> ../CSV/result_ICMP.out`)
                fi
        fi
done
```

그림 8. item 수에 따른 필터링
Fig. 8. Filtering by the number of items

이후 Fig 8과 같이 삭제된 Rule 파일에서 각 Rule이 가지고 있는 필드(아이템)의 개수를 파악한다. Rule 파일은 Fig 9와 같이 기본적으로 아이템이 많은 Rule이 위쪽에 아이템이 적은 Rule이 아래쪽에 분포되므로, k번째 Rule보다 k+1번째 Rule의 아이템수가 많다면 k번째 Rule이 그 위의 k-1, k-2, k-3 … 의 Rule을 포함한다고 할 수 있다. 그런 방식으로 모든 Rule에서 k번째 Rule을 찾아 최종적으로 사용할 Rule을 남긴다.

```
Attack=NO <- SPT=32976,LEN=40,MAC=00:00:00:00:00:08,PROTO=UDP,OUT= (12.5, 100)
Attack=NO <- SPT=32976,LEN=40,MAC=00:00:00:00:00:08,PROTO=UDP (12.5, 100)
Attack=NO <- SPT=32976,IN=lo,MAC=00:00:00:00:00:08,PROTO=UDP,OUT= (12.5, 100)
Attack=NO <- SPT=32976,IN=lo,MAC=00:00:00:00:00:08,PROTO=UDP (12.5, 100)
Attack=NO <- SPT=32976,MAC=00:00:00:00:00:08,PROTO=UDP,OUT= (12.5, 100)
Attack=NO <- SPT=32976,MAC=00:00:00:00:00:08,PROTO=UDP (12.5, 100)
Attack=NO <- DST=192.168.3.20,SPT=40949,DPT=53,TTL=64,LEN=40,IN=lo,MAC=00:00:00:00:00:08,PROTO=UDP,OUT= (15.625, 100)
Attack=NO <- DST=192.168.3.20,SPT=40949,DPT=53,TTL=64,LEN=40,IN=lo,MAC=00:00:00:00:00:08,PROTO=UDP (15.625, 100)
Attack=NO <- DST=192.168.3.20,SPT=40949,DPT=53,TTL=64,LEN=40,MAC=00:00:00:00:00:08,PROTO=UDP,OUT= (15.625, 100)
Attack=NO <- DST=192.168.3.20,SPT=40949,DPT=53,TTL=64,LEN=40,MAC=00:00:00:00:00:08,PROTO=UDP (15.625, 100)
Attack=NO <- DST=192.168.3.20,SPT=40949,DPT=53,TTL=64,IN=lo,MAC=00:00:00:00:00:08,PROTO=UDP,OUT= (15.625, 100)
Attack=NO <- DST=192.168.3.20,SPT=40949,DPT=53,TTL=64,IN=lo,MAC=00:00:00:00:00:08,PROTO=UDP (15.625, 100)
```

그림 9. 룰 파일
Fig. 9. Rule file

마. Create Policy Module 구현

알고리즘 동작 후 생성된 Rules는 IPtables 정책 형식과 다르기 때문에 적용이 불가능하다. 따라서 생성된 Rules를 IPtables에 적용하기 위한 문자열 Parsing 작업이 필요하다.

화이트리스트 기반으로 네트워크 공격이 발생했을 때에도 정상으로 판단된 패킷들은 접근이 허용되는 정책을 수립한다. 하지만 정상 패킷으로 판단되었더라도 일정 시간 동안 일정 수 이상의 패킷을 전송한다면 DROP 시켜야 한다.

```
                output = fopen("tcpTemp.rules", "a");

                if (mac[0] != 0 && source[0] != 0 && protocol[0] != 0 && port[0] != 0)
                {
                        fprintf(output, "-A INPUT -s %s -p %s --sport %s -m mac --mac %s -m recent --update --seconds 10 --hitcount 10 -j DROP\n", source, protocol, port, mac);
                        fprintf(output, "-A INPUT -s %s -p %s --sport %s -m mac --mac %s -j ACCEPT\n", source, protocol, port, mac);
                }
                else if (mac[0] != 0 && source[0] != 0 && protocol[0] != 0)
                {
                        fprintf(output, "-A INPUT -s %s -p %s -m mac --mac %s -m recent --update --seconds 10 --hitcount 10 -j DROP\n", source, protocol, mac);
                        fprintf(output, "-A INPUT -s %s -p %s -m mac --mac %s -j ACCEPT\n", source, protocol, mac);
                }
```

그림 10. IPtables 형식을 위한 룰 파싱
Fig. 10. Parsing the rules for IPtables format

위 Fig 10에서는 알고리즘 동작 후 생성된 정책을 문자열 Parsing 후 ACCEPT하는 정책과 정상 패킷으로 판단되었더라도 10초에 10번 이상 접근을 시도하면 DROP시키는 정책을 IPtables의 정책 형태에 맞게 수정한다. 여기서 중요한 것은 DROP 정책이 ACCEPT 정책의 상단에 위치해야 한다는 것이다.

생성된 정책을 방화벽에 적용하기 위한 작업으로, 아래 Fig 11에서는 IPtables에 적용시키기 위한 파일(iptables_protocol.rules)을 생성하고 테이블 설정, ULOG 그룹 설정 등을 수행한다.

```
        result = fopen("../result/iptables_total.rules", "r");

        if(result == NULL)
        {
                system("echo \"*mangle\" >> ../result/iptables_
total.rules");
                system("echo \"-A PREROUTING -j NFLOG --nflog-g
roup 0 \" >> ../result/iptables_total.rules");
                system("echo \"-A POSTROUTING -j NFLOG --nflog-
group 1\" >> ../result/iptables_total.rules");
        }


        system("sort -u tcpTemp.rules >> ../result/iptables_tot
al.rules");

        fclose(result);
        system("rm tcpTemp.rules");
        system("rm ../CSV/result_TCP.out");
        return 0;
```

그림 11. IPtables에 룰 적용
Fig. 11. Applying the rules to IPtables

Ⅳ. 구현 테스트

1. 성능 평가

테스트를 진행하기 위한 환경은 아래의 Table 4와 같다.

표 4. 테스트 환경
Table 4. Test environment

Role	Attacker	User	Firewall
OS	Ubuntu 16.04	Windows 7	Ubuntu 16.04
Tool	Hping 3.8	nping	-

가. 테스트

본 연구의 테스트 과정은 아래의 2가지로 나뉜다.

먼저 아래 Fig 12과 같이 네트워크 공격이 없는 상황에서 정상 사용자가 접속하여, 개발한 Modules을 통해 자동으로 방화벽 규칙이 생성/적용되도록 한다.

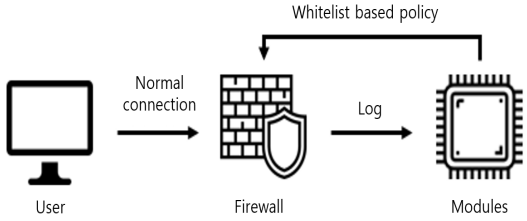

그림 12. 사용자 접속 학습
Fig. 12. User connection training

이후 아래 Fig 13과 같이 네트워크 공격을 발생시킨다. 이 때, 공격은 차단하면서 학습된 정상 접속은 가능함을 확인한다.

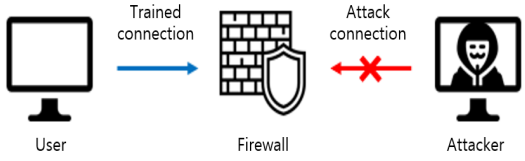

그림 13. 테스트
Fig. 13. Testing

나. 결과 분석

(1) 정상 패킷 학습

정상 사용자 PC로 방화벽 PC에 Protocol별 패킷을 8개씩 전송했다. 정상 사용자 PC로부터 패킷을 받은 방화벽 PC에서는 개발한 모듈들이 순차적으로 실행되고 학습 과정을 마친 후 Fig 14와 같이 IPtables 정책 Format에 맞는 형태로 정책을 생성 후 적용되는 것을 확인할 수 있었다.

```
-A INPUT -s 127.0.0.1 -p UDP -m mac --mac 00:00:00:00:00:08 -j
ACCEPT
-A INPUT -s 127.0.0.1 -p UDP -m mac --mac 00:00:00:00:00:08 -m
recent --update --seconds 10 --hitcount 10 -j DROP
-A INPUT -s 127.0.0.1 -p UDP --sport 246 -j ACCEPT
-A INPUT -s 127.0.0.1 -p UDP --sport 246 -m recent --update --s
econds 10 --hitcount 10 -j DROP
-A INPUT -s 127.0.0.1 -p UDP --sport 648 -j ACCEPT
-A INPUT -s 127.0.0.1 -p UDP --sport 648 -m recent --update --s
econds 10 --hitcount 10 -j DROP
-A INPUT -s 127.0.0.1 -p UDP --sport 976 -j ACCEPT
-A INPUT -s 127.0.0.1 -p UDP --sport 976 -m recent --update --s
econds 10 --hitcount 10 -j DROP
-A INPUT -s 127.0.0.1 --sport 246 -m mac --mac 00:00:00:00:00:0
8 -j ACCEPT
-A INPUT -s 127.0.0.1 --sport 246 -m mac --mac 00:00:00:00:00:0
8 -m recent --update --seconds 10 --hitcount 10 -j DROP
-A INPUT -s 127.0.0.1 --sport 648 -m mac --mac 00:00:00:00:00:0
8 -j ACCEPT
```

그림 14. 생성된 IPtables 정책
Fig. 14. Generated IPtables policies

(1) 네트워크 공격과 학습된 패킷의 동시 접속

공격자 PC로 SYN Flooding 공격을 수행함과 동시에 정상 사용자 PC로 접속을 시도했다. 그 결과 Fig 15와 같이 공격자 PC에서 총 33997개의 패킷을 전송했지만 그 중 98%의 패킷이 차단되는 것을 확인할 수 있었다. 차단 기준을 시간당 패킷량을 기준으로 하였기 때문에 전체 패킷 100%중 차단되지 않은 나머지 2%는 단위 시간당 차단 패킷량 기준치에 해당하기 이전에 보내진 것이다. 차단되지 않은 공격으로 인해 발생되는 서버의 부하는 적기 때문에 서버는 정상적으로 동작한다.

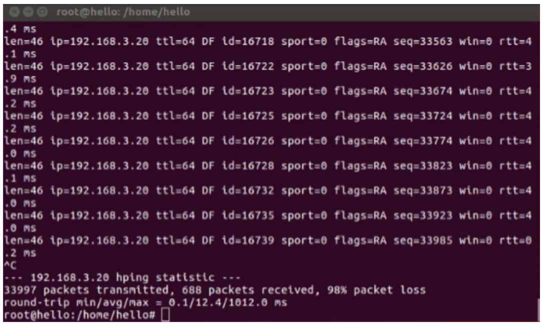

그림 15. 공격 결과(98% 차단 됨)
Fig. 15. Attack result(98% dropped)

반면 Fig 16과 같이 정상 PC에서 전송한 14개의 패킷은 학습이 되어있기 때문에 모두 허용됨을 확인할 수 있었다. 따라서 정상 패킷은 통과시키면서 공격을 차단하였음을 확인할 수 있다.

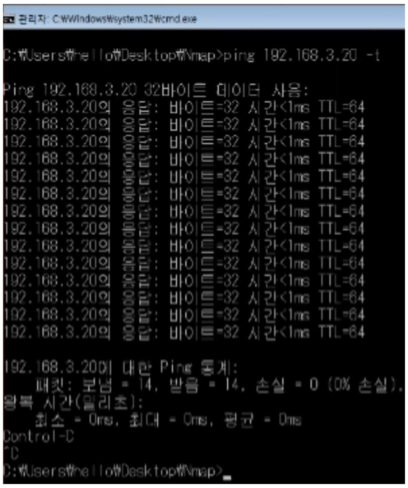

그림 16. 정상 접속 100% 허용
Fig. 16. Normal connection accepted 100%

V. 결 론

본 논문에서 제안하는 시스템을 통해 기존에 발생하던 다양한 문제들을 해결할 수 있다. 먼저, 정책을 수동으로 입력하던 방식에서 자동으로 생성/적용함으로써 보안 인력 및 비용의 감소를 기대할 수 있다. 또한, 정상 패킷의 학습과 방화벽 정책의 생성 및 적용하는 일련의 과정이 자동으로 이루어지기 때문에 보안 전문가가 아니더라도 사용할 수 있다. Rule Reduction Module의 개발 및 적용으로 Apriori 알고리즘 사용 시 발생할 수 있는 대량의 Rules가 생성되는 문제점을 해결하여 방화벽의 과부하 문제를 해결할 수 있다. 마지막으로, DoS 공격과 같은 불특정 다수의 공격이 발생 하더라도 학습된 정상 접속자에게는 접속을 허용하여 기존 방화벽의 문제점이었던 네트워크 공격 발생 시 공격 패킷과 정상 패킷을 모두 차단하는 문제점을 해결할 수 있다.

향후에는 미탐의 발생 가능성을 최소화하기 위해 Text Mining을 이용해 공격 패턴을 분석하고, Black List 정책을 자동으로 생성, 적용하는 방안을 연구할 것이다. 그리고 패킷의 Body 부분을 로그로 남겨 정책에 적용하여 네트워크 계층뿐만 아니라 세션, 연결 및 상위 계층에서의 다양한 공격까지 대응하기 위한 방안을 연구할 예정이다.

References

[1] http://www.etnews.com/20170626000221

[2] L. Ertoz E. Eilertson, A. Lazarevic, P. N. Tan, V. Kumar, J. Srivastava and P. Dokas, "MINDS - Minnesota Intrusion Detection System," Next Generation Data Mining, MIT Press, 2004.

[3] Young-seok Lee, "DDoS Attack Response Framework using Mobile Code", The Journal of Korea Institute of Information, Electronics, and Communication Technology, Vol. 3, No. 3, pp. 31-38, 2010. DOI: https://doi.org/10.17661/JIIECT.2017.11.30.

[4] N. Khamphakdee, N. Benjamas and S. Saiyod, "Improving Intrusion Detection System Based on Snort Rules for Network Probe Attacks Detection with Association Rules Technique of Data Mining," Journal of ICT Research and Applications, Vol. 8, No. 3, pp. 234-250, 2015.

[5] Kyu-Seok Shim, Sung-Ho Yoon, Su-Kang Lee and Myung-Sup Kim, "SigBox: Automatic Signature Generation Method for Fine-Grained Traffic Identification," Journal of Information Science and

Engineering, Vol. 33, pp. 537-569, 2017.

[6] Kim, Hyun-Seok, and Dong-Gue Park, "Implementation of abnormal behavior detection system based packet analysis for industrial control system security.", Journal of the Korea Academia-Industrial cooperation Society, Vol. 19, No. 4, pp. 47-56, 2018.

[7] Sang-Yong Choi, Eun-Young Cheon, and Dae-Sik Ko, "Analysis of the Network Traffic Collection Performance Using Suricata", Journal of KIIT, Vol. 17, No. 8, pp. 59-66, August. 31, 2019.

[8] Rakesh Agrawal and Ramakrishnan Srikant, "Fast Algorithms for Mining Association Rules", Proceedings of the 20th VLDB Conference, September. 12-15, 1994.

[9] Christian Borgelt and Rudolf Kruse, "Induction of Association Rules: Apriori Implementation", Compstat, Physica, Heidelberg, 395-400, 2002.

저 자 소 개

한 경 현(정회원)

- 2015년 2월 : 홍익대학교 컴퓨터정보통신학과 (학사)
- 2017년 2월 : 홍익대학교 전자전산공학과 (석사)
- 2017년 3월 ~ 현재 : 홍익대학교 전자전산공학과 (박사)

황 성 운(정회원)

- 1993 : 서울대 수학과 (학사)
- 1998 : 포항공과대학교 정보통신학과 (석사)
- 2004 : 한국과학기술원 전자전산학과 (박사)
- 1998 1월~ 2008 2월: 한국전자통신연구원 선임연구원
- 2008 3월~ 2020 2월: 홍익대학교 소프트웨어융합학과 교수
- 2020 3월~ 현재 : 가천대학교 컴퓨터공학과 교수