네트워크 공격 분석 및 트래픽 시각화 (aDDoS)

팀 명 :	늙었조
지 도 교 수 :	이병천 교수님
팀 장 :	이두리
팀 원 :	박주형
	송경선
	이라규
	이건우
	김동철

2024. 11.

중부대학교 정보보호학과

목 차

1. 서론

1.1 연구 배경

현대 사회에서는 정보통신기술의 급속한 발전과 함께 다양한 네트워크 기반 서비스가 증가하고 있으며, 이러한 서비스들은 일상생활과 비즈니스 전반에서 중요한 역할을 하고 있다. 그러나 네트워크 환경이 발전함에 따라 사이버 공격의 위협 또한 증가하고 있다.

네트워크 공격은 대규모 트래픽을 유발하거나 취약점을 악용하여 시스템을 무력화시키고 정상적인 서비스 제공을 방해하는 행위로, 이러한 공격으로 인한 서비스 중단은 기업에 막대한 경제적 손실을 초래할 뿐만 아니라 고객 신뢰도 하락 등의 문제를 일으킬 수 있다. 이에 따라 네트워크 공격에 효과적으로 대응할 수 있는 기술 개발은 중요한 과제가 되었다.

1.2 연구 목적 및 필요성

본 연구의 주된 목적은 네트워크 공격을 심층적으로 분석하고, 이를 통해 향후 유사한 공격에 효과적으로 대비하여 전반적인 사이버 보안 환경을 강화하는 데 있다. 최근 구독형이나 보안 관련 상품들이 많지만, 우리는 네트워크 공격에 대한 솔루션을 개발하여 네트워크 장비 없이 소프트웨어로 해결할 수 있는 보안 시스템에 기여하고자 한다.

이를 위해 ELK Stack을 기반으로 한 미러링 서버를 구축하여 네트워크 트래픽을 분석할 수 있는 기능을 갖추고 있으며, 트래픽과 로그 정보를 효과적으로 시각화하여 웹 기반 대시보드를 통해 사용자들에게 보다 직관적이고 보기 편한 데이터를 제공하고자 한다. 이러한 시각화 작업을 통해 사용자들이 중요한 정보를 쉽게 파악할 수 있도록 돕는 것이 목표이다.

또한 CICDDoS 2019 데이터셋을 활용하여 DDoS 공격 감지 모델을 학습시키고, 다양한 IPS/IDS 규칙을 통해 시스템 보안을 강화하고자 한다. 본 연구를 통해 네트워크 분석 및 모니터링 시스템을 체계적으로 구축함으로써, 기업 및 개인의 네트워크 자산을 보다 효과적으로 보호할 수 있을 것으로 기대된다.

궁극적으로 이 시스템은 사이버 공격으로 인한 피해를 최소화하고, 안전한 네트워크 환경을 유지하는 데 기여할 것이다. 이를 통해 조직은 운영 효율성을 높이고, 고객 신뢰를 구축하며, 사이버 보안 위협에 대한 저항력을 강화할 수 있을 것이다.

2. 관련연구

2.1 Python

Python은 고수준의 범용 프로그래밍 언어로, 간결하고 가독성이 높은 문법을 제공한다. 이 언어는 1991년에 귀도 반 로썸(Guido van Rossum)에 의해 처음 개발되었으며, 이후 지속적으로 발전해 왔다. Python의 가장 큰 장점 중 하나는 다양한 라이브러리와 프레임워크가 존재하여 웹 개발, 데이터 분석, 머신러닝 등 여러 분야에서 폭넓게 사용될 수 있다는 점이다. 특히 Pandas, NumPy, Matplotlib와 같은 라이브러리를 통해 데이터 과학 분야에서 강력한 도구로 자리 잡았다.

2.2 Django

Django는 Python으로 작성된 고급 웹 프레임워크로, 신속한 개발과 간편한 유지보수를 목적으로 설계되었다. 이 프레임워크는 MVC(Model-View-Controller) 아키텍처를 따르며, ORM(Object-Relational Mapping) 기능을 통해 데이터베이스 작업을 직관적으로 처리할 수 있게 해준다. Django의 강력한 보안 기능은 SQL 인젝션, 크로스 사이트 스크립팅(XSS), 크로스 사이트 요청 위조(CSRF)와 같은 일반적인 보안 위협으로부터 애플리케이션을 보호하는 데 중요한 역할을 한다.

2.3 MongoDB

MongoDB는 NoSQL 데이터베이스로, 비정형 데이터의 저장과 관리를 지원하는 혁신적인 시스템이다. MongoDB는 JSON과 유사한 BSON(Binary JSON) 형식으로 데이터를 저장하여, 유연한 데이터 구조를 제공한다. 이로 인해 사용자는 데이터를 쉽게 모델링하고, 복잡한 쿼리도 간단하게 처리할 수 있다. MongoDB의 데이터베이스는 수평 확장이 용이해 대량의 데이터를 효율적으로 처리할 수 있으며, 고성능을 요구하는 애플리케이션에 적합하다.

2.4 Ubuntu

Ubuntu는 다양한 소프트웨어와 패키지를 쉽게 설치할 수 있는 APT(Advanced Package Tool) 패키지 관리 시스템을 갖추고 있으며, 이를 통해 사용자는 필요한 응용 프로그램을 신속하게 설치하고 관리할 수 있다. 초보자부터 전문가까지 폭넓은 사용자층을 아우르는 운영체제로, 클라우드 컴퓨팅, 서버 관리 및 개발 환경에 많이 사용된다.

2.5 Suricata

Suricata는 멀티스레딩을 지원하는 고성능 네트워크 IDS 및 IPS로, 패킷 캡처, 프로토콜 분석, 침입 탐지 기능을 제공하는 오픈 소스 소프트웨어이다. 사용자 정의 탐지 규칙을 설정할 수 있는 YAML 형식의 규칙 파일과 다양한 로그 포맷을 지원하여 모니터링 및 분석을 용이하게 하며, HTTP, FTP, DNS 등 여러 프로토콜을 깊이 있게 분석해 의심스러운 트래픽을 식별할 수 있다. 또한 PCAP 형식의 패킷 캡처 파일을 처리하여 과거의 트래픽 분석이 가능하며, 이를 통해 보안 팀은 사이버 공격을 탐지하고 대응할 수 있다.

2.6 Kafka

Kafka는 대규모 데이터 스트리밍 플랫폼으로, 분산형 메시징 시스템을 기반으로 한다. 고속의 데이터 전송과 처리 능력을 갖추고 있으며, 실시간 데이터 파이프라인 구축에 적합하다. Kafka는 발행-구독 모델을 사용하여, 다양한 데이터 소스에서 수집한 메시지를 중앙에서 관리하고 필요에 따라 소비자에게 전달한다. 이 시스템은 높은 내구성과 확장성을 제공하여, 대량의 데이터를 안정적으로 처리할 수 있는 장점이 있다.

2.7 Elasticsearch

Elasticsearch는 분산형 검색 및 분석 엔진으로, 대량의 데이터를 실시간으로 검색하고 분석하는 데 최적화되어 있다. RESTful API를 통해 다양한 데이터 소스와 쉽게 연동할 수 있으며, 강력한 쿼리 기능과 분석 기능을 제공한다. 특히, 텍스트 기반의 데이터 검색에 강점을 가지며, 로그 및 메트릭 데이터를 처리하는 데 널리 사용된다. Elasticsearch는 클러스터링 기능을 통해 데이터의 가용성과 신뢰성을 높이며, 비즈니스 인사이트를 도출하는 데 중요한 역할을 한다.

2.8 Filebeat

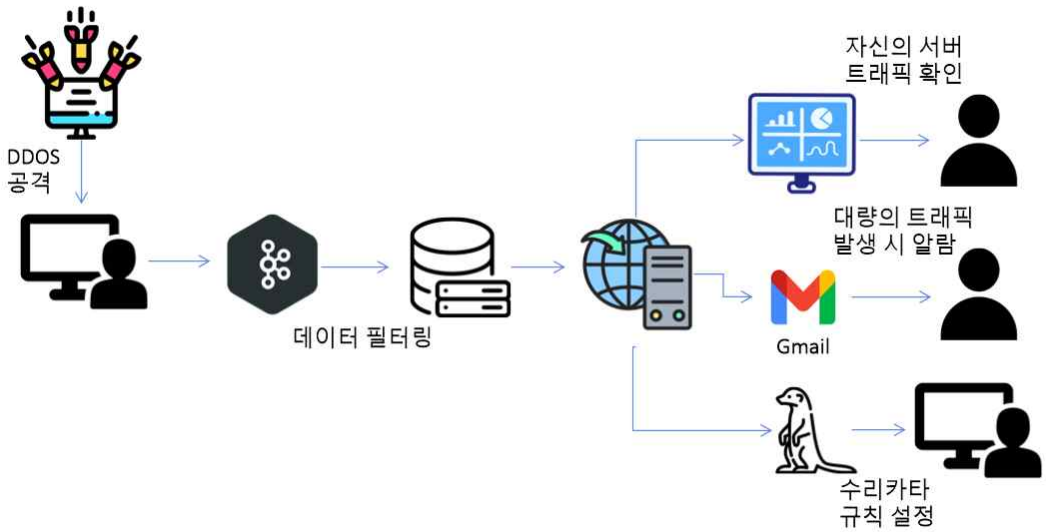
Filebeat는 경량의 데이터 수집기로, 파일 로그를 수집하여 Elasticsearch 또는 Logstash로 전송하는 역할을 한다. 다양한 로그 파일 형식을 지원하며, 설정이 간단하여 빠르게 배포할 수 있다. Filebeat는 비동기식으로 로그 데이터를 수집하고, 중복 방지 및 데이터 유실 방지 기능을 통해 안정적인 데이터 전송을 보장한다. 이 도구는 특히 로그 모니터링 및 분석을 위한 기반을 마련해 주며, DevOps 환경에서 필수적인 요소로 자리 잡고 있다.

3. 본론

3.1 서비스 구상

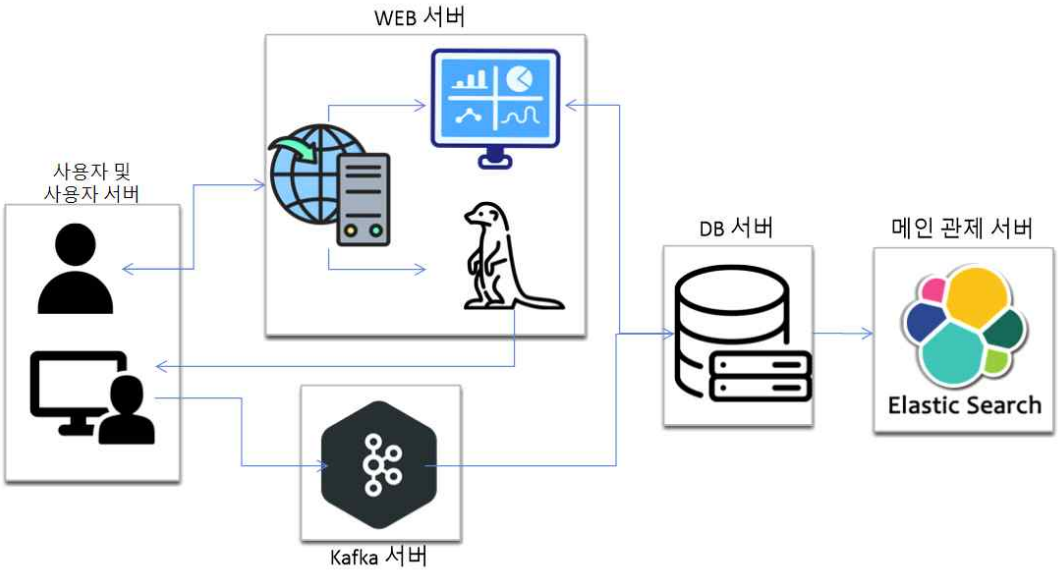
[그림 1]은 외부로부터의 네트워크 공격을 감지하는 과정을 보여준다. 먼저, 서버에 설치된 Suricata는 네트워크 패킷을 분석하고, 의심스러운 트래픽을 필터링하여 로그 파일로 저장한다. 이때, 트래픽 정보는 Suricata를 통해 지속적으로 수집되며, 데이터를 Filebeat를 이용해 카프카 서버로 전송한다. Filebeat는 Kafka와 같은 데이터 처리 플랫폼을 사용하여 트래픽 데이터를 전달하고, 이 데이터를 기반으로 분석이 이루어진다.

Kafka에 저장된 데이터는 MongoDB에 전송되어, 저장되고, 필요에 따라 분석 및 시각화 도구(ELK Stack)로 전송되어 분석이 가능하게 한다. 비정상적인 트래픽이 감지되면, 관리자는 Gmail을 통해 경고를 받고, 이를 확인하여 신속하게 대응할 수 있게 할 수 있다. 또한, Suricata의 규칙을 수정함으로써 향후 동일한 유형의 공격에 대해 효과적으로 방어할 수 있다.



[그림 1] 서비스 구상도

[그림 2]의 메인 관제 서버는 MongoDB에 저장된 데이터를 분석한다. MongoDB에 새롭게 추가된 트래픽 데이터를 Filebeat를 통해 ELK가 적용된 메인 관제 서버로 전송하고, 이를 기반으로 데이터를 분석하여 시각화한다. 이 분석을 통해 비정상적인 트래픽이나 네트워크 공격을 감지할 수 있으며, 이를 통해 현재 어떤 방식으로 네트워크 공격이 이루어지는지 분석하고 대응 매뉴얼을 만들어 사용자들에게 제공한다. 이와 같은 시스템 구성을 통해 사용자 서버에서 발생하는 트래픽을 모니터링하고, 데이터가 Kafka와 MongoDB를 통해 메인 관제 서버로 전달되어 ELK 스택으로 분석되는 전체적인 흐름을 구현할 수 있다.



[그림 2] 아키텍처 구상도

3.2 개발 환경

분류	도구/기술 스택
프로그래밍 언어 및 프레임워크	Python, Django
데이터베이스	MongoDB
운영체제 및 환경 설정	Ubuntu, Git, Visual Studio Code
도구	ElasticSearch, Filebeat, Kibana, Suricata, Kafka

[표 1] 개발환경

3.3 서버 구성

3.3.1 피해자 서버

Suricata 설정을 자동화하고 관리하기 위해 설계되었다. 설정 파일과 룰 파일의 경로를 변수로 지정하여 Suricata의 설정 파일(suricata.yaml)과 룰 파일(suricata.rules)을 쉽게 참조할 수 있도록 한다.

```
Suricata_Cf=/etc/suricata/suricata.yaml
Suricata_Rules=/etc/suricata/suricata.rules
```

[그림 3] 설정 파일(suricata.yaml)과 룰파일(suricata.rules) 변수

WELLPLAY 함수는 명령어가 제대로 실행되었는지 확인하고, 성공 여부를 출력한다. 명령이 정상적으로 수행되면 “[OK]” 메시지를 출력하고, 실패 시 “[FAIL]” 메시지와 함께 로그 확인을 안내한다.

START 함수는 Suricata 서비스를 재시작하고 변경된 설정을 적용하는 역할을 한다. 설정이 정상적으로 적용되면 서비스를 활성화하고 상태를 확인하며, 오류가 발생하면 적절한 메시지를 출력한다. 이러한 구조는 설정 변경 후 Suricata가 정상적으로 작동하는지 확인하고 관리하는데 중요한 기능을 제공한다.

```
# Function

WELLPLAY() {
  job="$1"
  if [ $? -eq 0 ]; then
    echo "[ OK ] $job 완료"
  else
    echo "[FAIL] $job 에 실패하였습니다. 로그를 확인해 보세요"
    exit 1
  fi
}

START() {
  SVC="$1"
  systemctl restart $SVC
  if [ $? -eq 0 ]; then
    systemctl enable $SVC
    systemctl status $SVC
  else
    echo "[FAIL] 서비스가 켜지지 않았습니다. 로그를 확인해 보세요"
    exit 1
  fi
}
```

[그림 4] WELLPLAY 함수와 START 함수

Suricata 설치는 패키지 저장소를 추가한 후 업데이트와 함께 패키지를 설치하는 방식으로 진행된다. WELLPLAY 함수는 설치가 성공적으로 완료되었는지 확인한다.

```
### 수리카타 ###
# 1. 수리카타 설치
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt-get update
sudo apt-get install -y  -q suricata
WELLPLAY Suricata설치
```

[그림 5] 수리카타 설치 명령어

설치가 완료된 후, Suricata의 로그 수집 방식을 수정한다. 기본적으로 fast.log는 비활성화되며, eve.json 파일로 로그를 수집하도록 설정을 변경한다. 다양한 로그 유형(alert, http, dns 등)을 포함하는 eve.log 설정을 적용한다.

```
# 2. 수리카타 설정파일 및 룰 설정
# fast -> disable
sed -i '85s/enabled: yes/enabled: false/' $Suricata_Cf
WELLPLAY fastlog설정

# eve.log
sed -i '94 a\
        types:\n\
          - alert\n\
          - http\n\
          - dns\n\
          - icmp\n\
          - smtp\n\
          - ssh\n\
          - fileinfo\n\
          - flow\n\
          - tls\n\
          - stats' $Suricata_Cf
WELLPLAY eve.log설정
```

[그림 6] 수리카타 설정 파일

이와 같은 과정으로, Suricata가 fast.log 대신 eve.json을 사용하여 다양한 유형의 로그를 수집할 수 있게 설정된다.

Suricata에서 사용할 규칙 파일을 생성하고 적절한 권한을 부여한다. 이후 DDoS 공격, TCP 및 UDP Flood, HTTP Flood, DNS Flood, ICMP Flood 등을 탐지할 수 있는 규칙을 추가한다. 이 규칙들은 트래픽을 모니터링하면서 이상 징후를 탐지하는 데 사용된다.

```
# suricata-rules
touch $Suricata_Rules
chmod u+w $Suricata_Rules

cat << EOF > $Suricata_Rules
alert udp any any -> any 53 (msg:"Possible DNS DDoS Attack Detected"; threshold: type both, track by_src, count 100, seconds 10; fl
alert icmp \$EXTERNAL_NET any -> \$HOME_NET any (msg:"ICMP Echo Request Flood Detected"; icmp_type:8; threshold:type:threshold, tra
alert tcp any any -> any 80 (flags: S; flow: stateless; threshold: type both, track by_src, count 100, seconds 10; msg: "Possible S
alert udp any any -> any any (msg:"Possible UDP Flood Detected"; threshold: type both, track by_src, count 1000, seconds 10; sid:10
alert http any any -> any 80 (msg:"Possible HTTP Flood Detected"; flow: established; threshold: type both, track by_src, count 100,
EOF
WELLPLAY rules파일생성
```

[그림 7] 수리카타 룰 설정

Suricata가 네트워크 트래픽을 감지할 수 있도록 네트워크 인터페이스를 설정한다. eth0 인터페이스를 ens33으로 변경하여 Suricata가 해당 인터페이스를 통해 트래픽을 분석하도록 구성한다. 설정 후 Suricata 서비스를 재시작하여 변경 사항을 적용한다.

```
# af-packet
sed -i 's/interface: eth0/interface: ens33/' $Suricata_Cf
WELLPLAY 네트워크인터페이스설정

# 3. 수리카타 재시작 및 enable 설정
START suricata.service
WELLPLAY 서비스시작
```

[그림 8] 네트워크 인터페이스 설정

Filebeat는 공식 Elastic 사이트에서 다운로드하여 설치할 수 있다. 아래 명령어를 사용해 다운로드한 후, WELLPLAY 함수를 사용해 설치 성공 여부를 확인한다.

```
### 파일비트 ###
# 1. 파일비트 설치
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-8.10.1-amd64.deb
sudo dpkg -i filebeat-8.10.1-amd64.deb
WELLPLAY 파일비트설치
```

[그림 9] Filebeat 설치 명령어

Filebeat가 Suricata에서 생성된 eve.json 로그 파일을 수집하도록 설정한다. 설정 파일의 input 섹션에서 로그 파일 경로를 /var/log/suricata/eve.json으로 지정한다. 이후, 수집된 로그 데이터를 카프카 서버로 전송하기 위해 output.kafka 설정을 추가하여 호스트 정보와 토픽을 설정한다.

```
# 2. 파일비트 설정 파일 수정
# input 설정
sed -i '/^filebeat.inputs:/a \
- type: log\
  enabled: true\
  paths:\
    - /var/log/suricata/eve.json
' $Filebeat_Cf
WELLPLAY Input설정완료

# output설정
sed -i 's/^output.elasticsearch:/# output.elasticsearch:/' $Filebeat_Cf
WELLPLAY Output.elasticsearch설정

sed -i '$a \
output.kafka:\
  hosts: ["10.100.114.199:9092"]\
  topic: "kafka"\
  partition.round_robin:\
    reachable_only: false\
  required_acks: 1' $Filebeat_Cf
WELLPLAY kafka연결
```

[그림 10] input 설정 및 output 설정

설정이 완료되면 Filebeat 서비스를 재시작하고 활성화하여 설정한 내용을 적용한다.

```
# 파일비트 재시작 및 enable설정
START filebeat.service
WELLPLAY 서비스시작
```

[그림 11] Filebeat 재시작 명령어

메일 알림을 위한 주요 변수를 정의한다. 여기에는 Suricata의 eve.json 로그 파일 경로, 알림 임계값(1MB), 체크 주기(60초), 그리고 알림을 받을 이메일 주소가 포함된다.

```
# 이메일 관련 변수
EVE_LOG=/var/log/suricata/eve.json
ALERT_THRESHOLD=1048576 # 1MB
CHECK_INTERVAL=60 # 초 단위
EMAIL="$1" # 알림을 받을 이메일 주소
```

[그림 12] 이메일 알림 변수

두 가지 주요 함수를 사용하여 로그 파일을 모니터링하고, 트래픽이 설정된 임계값을 초과할 때 사용자에게 이메일로 알림을 보낸다.

√ send_alert : 사용자가 인자로 넘긴 이메일 주소로 알림을 전송하는 함수이다. 트래픽이 갑작스럽게 증가할 때 해당 이메일로 경고 메시지를 보낸다.

√ monitor_log_size : Suricata의 eve.json 파일 크기를 모니터링하는 함수로, 10초 동안 로그 파일 크기가 50MB 이상 증가하면 사용자에게 이메일을 보낸다.

```
send_alert() {
  if echo "많은 트래픽이 감지되었습니다. 저희 서비스를 방문해주세요" | msmtp --from=default -t "$EMAIL"; then
    echo "[ OK ] 메일이 성공적으로 발송되었습니다."
  else
    echo "[FAIL] 메일 발송에 실패했습니다."
  fi
}

monitor_log_size() {
  previous_size=0
  while true; do
    current_size=$(stat -c%s "$EVE_LOG")
    if [ $previous_size -gt 0 ]; then
      size_diff=$((current_size - previous_size))
      if [ $size_diff -gt $ALERT_THRESHOLD ]; then
        send_alert
      fi
    fi
    previous_size=$current_size
    sleep $CHECK_INTERVAL
  done
}
```

[그림 13] send_alert 함수와 monitor_log_size 함수

메일 전송을 위한 msmtp 패키지를 설치한다.

```
############## Mail Service #############

# msmtp 설치
sudo apt-get install -y -q msmtp
WELLPLAY msmtp설치
```

[그림 14] msmtp 패키지 설치 명령어

msmtp를 통해 이메일을 전송하기 위해 Google 계정 정보를 설정 파일에 등록한다. 이 설정 파일에는 Gmail SMTP 서버 정보와 계정의 로그인 정보를 포함하고 있다. 이렇게 설정된 정보를 통해 서버는 Google 계정으로 이메일을 보낼 수 있다.

이메일 설정 파일은 보안상 중요한 정보를 포함하므로, 권한을 제한하여 접근을 방지한다. chmod 600 명령어를 사용해 파일 권한을 설정하여 파일 소유자만 읽고 쓸 수 있도록 제한한다.

```
# 이메일 알림 모니터링 시작
monitor_log_size &
WELLPLAY 메일서비스
```

[그림 15] 이메일 알림 모니터링 시작 명령어

```
# 설정파일 아이디 등록
cat << EOF > ~/.msmtprc
# Gmail SMTP 설정
account default
host smtp.gmail.com
port 587
from
auth on
user
password
tls on
tls_starttls on
tls_certcheck off
logfile ~/.msmtp.log
EOF
WELLPLAY 설정완료

# 권한주기
chmod 600 ~/.msmtprc
```

[그림 16] 파일 권한 설정

3.3.2 카프카 서버

카프카 서버는 트래픽 로그에서 중요한 필드만을 추출하여 데이터를 처리한다. 이를 위해 filtered_logs_required_fields 배열을 사용하여 필요한 필드 목록을 사전에 정의한다. 타임스탬프, 출발지/목적지 IP, 프로토콜, TCP 플래그 등의 필드는 트래픽 데이터 분석에 있어 핵심적인 정보이다.

```
# 필요한 필드 정의
filtered_logs_required_fields = [
    '@timestamp', 'src_ip', 'src_port', 'dest_ip', 'dest_port', 'proto', 'tcp_flags', 'syn', 'ack',
    'state', 'reason', 'flow_id', 'app_proto', 'tcp_flags_ts', 'tcp_flags_tc', 'bytes_toserver',
    'pkts_toserver', 'bytes_toclient', 'pkts_toclient', 'start', 'end', 'age', 'http_method', 'url',
    'http_user_agent', 'status', 'length', 'http_content_type', 'query_type', 'query_name',
    'icmp_type', 'icmp_code', 'response_code', 'Network Segment', 'event_type', 'in_iface'
]
```

[그림 17] 필터링된 필드

데이터를 저장하는 과정에서는 클라이언트 연결과 데이터 관리가 핵심적인 역할을 한다. MongoDB URL을 사용해 클라이언트를 생성하고 서버와의 연결을 확인하기 위해 ping() 명령어를 사용한다. 이를 통해 MongoDB와의 연결 상태가 정상적인지 확인할 수 있으며, 연결이 성공하면 'network_catcherdatabase' 내의 'traffic' 컬렉션에 접근하여 데이터를 저장할 수 있다.

저장된 데이터는 MongoDB의 TTL(Time To Live) 기능을 활용하여 일정 기간만 보관되도록 설정된다. TTL 설정을 통해 데이터는 5일 동안만 저장되며, 그 이후에는 자동으로 삭제된다. 이는 create_index 메소드를 사용해 타임스탬프 필드를 기준으로 이루어지며, 이를 통해 데이터베이스의 저장 공간을 효율적으로 관리하고 불필요한 데이터가 축적되는 것을 방지할 수 있다.

```
# MongoDB Atlas 설정
mongo_uri =
try:
    client = MongoClient(mongo_uri, tls=True, tlsAllowInvalidCertificates=True, serverSelectionTimeoutMS=500
    # 서버 상태 확인
    client.admin.command('ping')
    print("MongoDB Atlas 연결 성공")
except ConnectionFailure as e:
    print("MongoDB Atlas 연결 실패: %s" % e)
    exit(1)

# 데이터베이스와 컬렉션 설정
db = client
collection = db

# TTL 인덱스 생성
collection.create_index([('@timestamp', 1)], expireAfterSeconds=5*24*60*60)
```

[그림 18] MongoDB Atlas 설정, 카프카 서버 상태확인, TTL(Time-To-Live) 설정

카프카 소비자(consumer)는 KafkaConsumer 클래스를 사용해 생성되며, 카프카 토픽에서 메시지를 읽고 처리하는 역할을 한다. 이를 위해 bootstrapservers 옵션으로 카프카 서버의 주소를 지정하고, group_id 옵션을 통해 소비자 그룹을 설정하여 여러 소비자가 메시지를 분산 처리할 수 있게 한다.

소비자가 처음 실행될 때 auto_offset_reset 옵션이 설정되어, earliest로 지정하면 가장 오래된 메시지부터 읽기 시작한다. 또한, enable_auto_commit 옵션을 통해 메시지를 읽은 후 자동으로 오프셋을 저장하여 중복 읽기를 방지한다. 이러한 설정을 통해 카프카 소비자는 데이터를 효율적으로 수집하고 처리하며, 여러 소비자가 동시에 동일한 토픽을 중복 없이 처리할 수 있다.

[그림 19] 카프카 설정 및 소비자 그룹 설정

수집된 데이터에서 필요한 필드만을 추출하여 필터링하는 역할을 한다. message 필드가 JSON 형식일 경우 이를 파싱 하여 원본 데이터와 결합하고, timestamp가 없을 시 추가한다. 그런 다음, required_fields 리스트에 정의된 필드만 선택적으로 추출하며, 중첩된 구조에서 필요한 데이터를 찾아내고 존재하지 않는 필드에는 None을 할당한다. 이 과정을 통해 데이터를 효율적으로 정제하고, 필요한 정보만 남긴다.

```python
def extract_filtered_data(data, required_fields):
    # 'message' 필드가 JSON 문자열인 경우, 이를 파싱하여 추가 필드를 포함한 딕셔너리 생성
    message_data = data.get('message')
    if message_data:
        try:
            message_json = json.loads(message_data)
            # 'message' 데이터와 주 데이터에서 필드 추출
            combined_data = {**data, **message_json}
            # 'message' 안의 'timestamp' 값을 '@timestamp'로 설정
            if 'timestamp' in message_json:
                combined_data['@timestamp'] = message_json['timestamp']
        except json.JSONDecodeError:
            # 'message' 필드가 JSON 형식이 아닐 경우 원본 데이터 반환
            combined_data = data
    else:
        combined_data = data

    # 각 필드의 값을 데이터에서 추출
    filtered_data = {}
    for field in required_fields:
        if field in combined_data:
            filtered_data[field] = combined_data[field]
        elif field in combined_data.get('tcp', {}):
            filtered_data[field] = combined_data['tcp'].get(field)
        elif field in combined_data.get('flow', {}):
            filtered_data[field] = combined_data['flow'].get(field)
        elif field in combined_data.get('http', {}):
            filtered_data[field] = combined_data['http'].get(field)
        elif field in combined_data.get('fileinfo', {}):
            filtered_data[field] = combined_data['fileinfo'].get(field)
        else:
            filtered_data[field] = None

    return filtered_data
```

[그림 20] 수집 데이터 필터링 로직

카프카에서 수신한 메시지를 처리한 후 MongoDB Atlas에 저장하는 과정을 보여준다. 수신된 메시지는 먼저 UTF-8로 디코딩되고, JSON 형식으로 파싱 된다. 그 후, 필터링 함수인 extract_filtered_data()를 통해 필요한 필드만 추출하여 데이터를 정제한다. 정제된 데이터는 MongoDB에 저장되며, 저장된 정보는 웹과 메인 서버에서 활용된다. 마지막으로, 작업이 완료되면 소비자와 MongoDB 클라이언트 연결을 안전하게 종료한다.

```
try:
    for msg in consumer:
        if msg.value:
            # 메시지 값을 문자열로 디코딩
            message_value = msg.value.decode('utf-8')
            logging.debug("수신한 메시지: %s", message_value)
            try:
                # JSON 데이터로 파싱
                data = json.loads(message_value)
                logging.debug("파싱된 JSON 데이터: %s", json.dumps(data, indent=2, ensure_ascii=False))

                # 필터링된 데이터 추출
                filtered_data = extract_filtered_data(data, filtered_logs_required_fields)
                logging.debug("추출된 데이터: %s", json.dumps(filtered_data, indent=2, ensure_ascii=False))

                # MongoDB에 필터링된 데이터 삽입
                collection.insert_one(filtered_data)

            except json.JSONDecodeError as e:
                logging.error("JSON 디코딩 오류: %s", e)
        else:
            logging.debug("빈 메시지 수신")

except KeyboardInterrupt:
    logging.info("Consumer 종료")
finally:
    consumer.close()
    client.close()
```

[그림 21] 필터링데이터 JSON 형식 변환, DB 저장

3.3.3 메인 관제 서버

MongoDB Atlas를 활용하여 데이터베이스 연결을 설정하고, 네트워크 트래픽 데이터를 수집하고 관리한다. 클라이언트는 'network_catcher_database'라는 데이터베이스에 연결되며, 그 안의 'traffic' 컬렉션에서 변화를 감지한다. 이를 통해 서버는 네트워크 트래픽에서 발생하는 변경 사항을 처리할 수 있다.

```
# MongoDB Atlas 클라이언트 설정
client = MongoClient
```

[그림 22] 메인관제에 대한 DB설정

데이터 수집 과정에서는 MongoDB의 Change Stream 기능을 활용하여 트래픽 데이터의 변경 사항을 모니터링한다. 이 Stream은 'traffic' 컬렉션에서 발생하는 모든 이벤트를 감시하고, 변경이 발생할 때마다 해당 이벤트를 수집하는 역할을 한다. 이를 통해 데이터의 흐름을 끊임없이 파악하며, 중요한 변경 사항을 빠르게 감지할 수 있다.

```
# Change Stream 설정
with client[                    ][        ].watch() as stream:
    for change in stream:
```

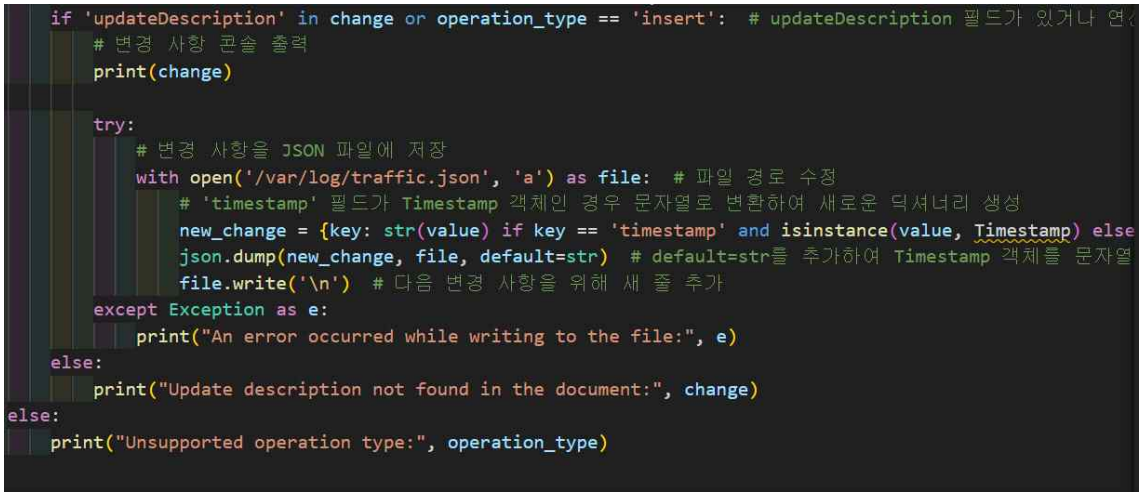
[그림 23] DB의 Change Stream 기능을 활용한 모니터링 실시

수집된 변경 사항 중에서는 'insert' 또는 'update' 작업만을 선별하여 처리한다. 이러한 작업 유형에 대한 필터링을 통해 중요한 데이터 변경만을 기록하고, 불필요한 데이터 처리의 부담을 줄일 수 있다. 필터링된 변경 사항은 로그 파일에 저장되며, 파일 형식은 JSON을 사용하여 구조화된 데이터를 저장한다. 로그 파일을 작성할 때는 오류가 발생할 가능성도 고려하여 예외 처리를 구현, 파일 저장 과정에서 문제가 발생하면 오류 메시지를 출력하여 문제를 신속히 해결할 수 있다.

```
if 'operationType' in change:  # 연산 유형이 있는지 확인
    operation_type = change['operationType']
    if operation_type == 'update' or operation_type == 'insert':  # 연산 유형이 update 또는 insert인 경우
```

[그림 24] 작업유형 필터링을 통한 불필요한 데이터 처리 제한

수집된 데이터는 Filebeat를 통해 ELK 스택과 연동된다. ELK는 수집된 네트워크 트래픽 데이터를 분석하고, 시각화하며, 관리자에게 직관적인 대시보드를 제공한다.



```
    if 'updateDescription' in change or operation_type == 'insert':  # updateDescription 필드가 있거나 연
        # 변경 사항 콘솔 출력
        print(change)

        try:
            # 변경 사항을 JSON 파일에 저장
            with open('/var/log/traffic.json', 'a') as file:  # 파일 경로 수정
                # 'timestamp' 필드가 Timestamp 객체인 경우 문자열로 변환하여 새로운 딕셔너리 생성
                new_change = {key: str(value) if key == 'timestamp' and isinstance(value, Timestamp) else
                json.dump(new_change, file, default=str)  # default=str를 추가하여 Timestamp 객체를 문자열
                file.write('\n')  # 다음 변경 사항을 위해 새 줄 추가
        except Exception as e:
            print("An error occurred while writing to the file:", e)
    else:
        print("Update description not found in the document:", change)
else:
    print("Unsupported operation type:", operation_type)
```

[그림 25] 데이터 변경 사항을 JSON 파일에 저장

이를 통해 현재 어떤 공격이 유행하는지 분석하여 대응 매뉴얼을 사용자들에게 제공할 수 있다.

4. 서비스 안내

4.1 Django 프로젝트 구성

Django 프로젝트는 하나의 큰 프로젝트 안에 여러 개의 앱을 관리하는 방식으로 구성된다. 프로젝트 루트에는 manage.py, settings.py, urls.py, asgi.py, wsgi.py가 존재하며 각각 웹 애플리케이션의 전반적인 설정과 구성을 관리한다. 이번 프로젝트에 한해서 asgi.py와 wsgi.py를 기본값으로 유지하였다.

[그림 26] Django 프로젝트, App 파일구조

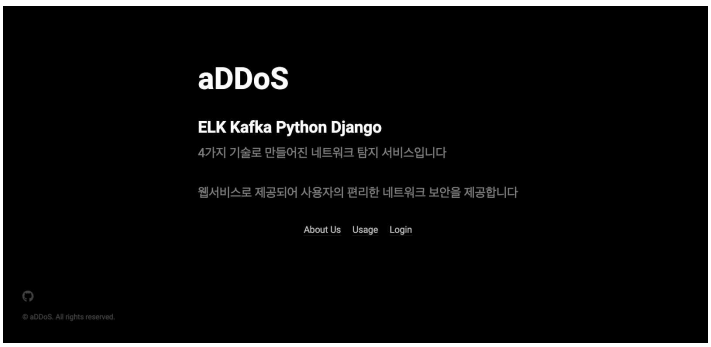
각각의 앱과 디렉터리는 다음과 같은 역할을 한다.

- home : 메인 페이지와 소개 페이지에 대한 구성요소
- mypage : Suricata 수정 페이지와 Traffic 페이지에 대한 구성요소
- user : 회원가입과 로그인 페이지, 기능에 관련된 구성요소
- static : CSS와 JS, 이미지와 같은 정적파일 요소
- templates : 각 앱별 HTML Template 구분

4.2 웹 페이지 구성

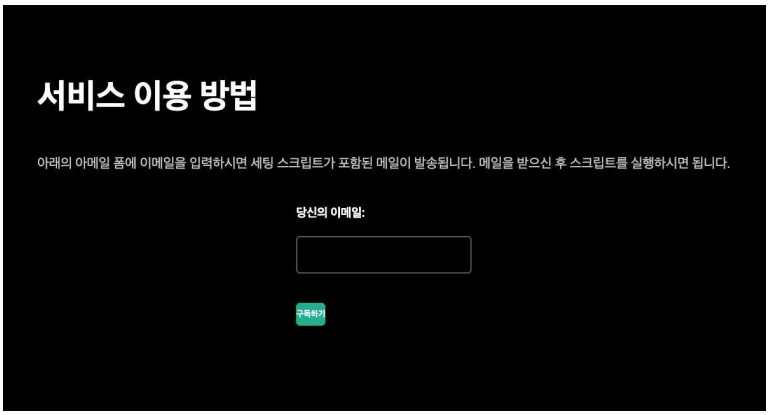
Django 프로젝트 안 template, static 디렉토리를 만들어 관리하였으며, 각각의 앱 안의 model.py, view.py, form.py를 활용,“{% load Static %}”을 통한 static 디렉터리의 정적파일을 Import 하여 해당 페이지를 구성, 기능을 구현하였다.

다음은 화면의 구성이다.



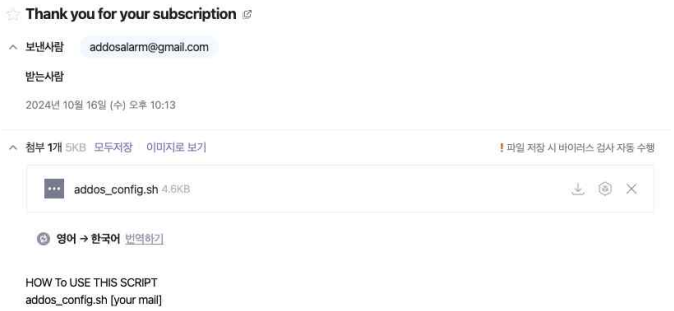
[그림 27] 서비스 초기페이지(메인페이지)

• 서비스를 접속했을 때 보이게 되는 첫 화면으로, 소개 페이지와 로그인에 접근할 수 있다.

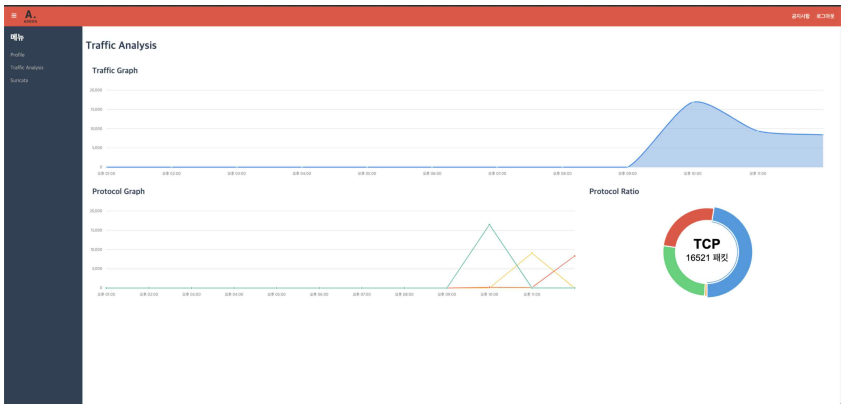


[그림 28] about 페이지 내부 사진(About us)

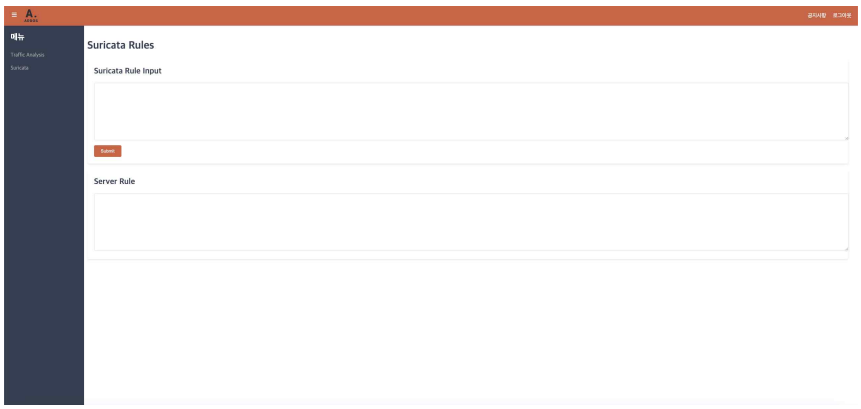
• 서비스의 소개 방법과 사용자 서버 설정 스크립트를 이메일로 받을 수 있다.



[그림 29] about 페이지를 통해 발송된 이메일



[그림 30] mypage/traffic

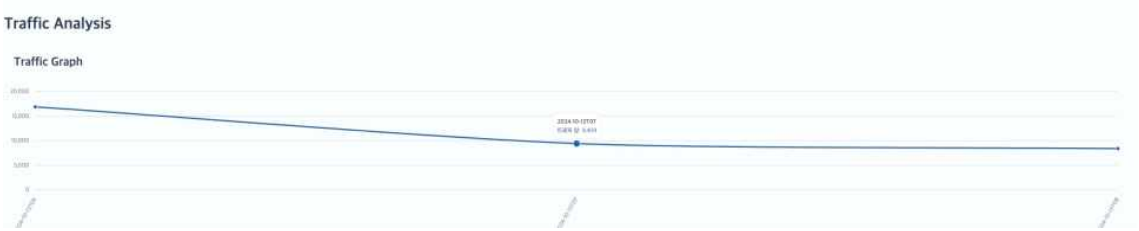


[그림 31] mypage/suricata

4.3 공격 대응 및 트래픽 분석

서비스에 있어 핵심 기능이자 **4.2**에 삽입한 mypage/traffic과 mypage/suricata를 세부 분석하여 정리하였다.

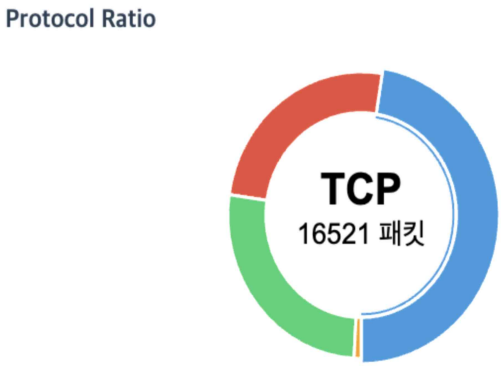
4.3.1 Traffic Graph



[그림 32] 트래픽량을 나타내는 그래프

위 그림은 DB에 저장된 자신의 서버에 온 트래픽 데이터들을 관점에 맞게 그래프로 가시화하여 사용자가 확인할 수 있다. 이를 통해 공격이 온 시점과 공격이 지속된 시간을 알고 대응할 수 있다.

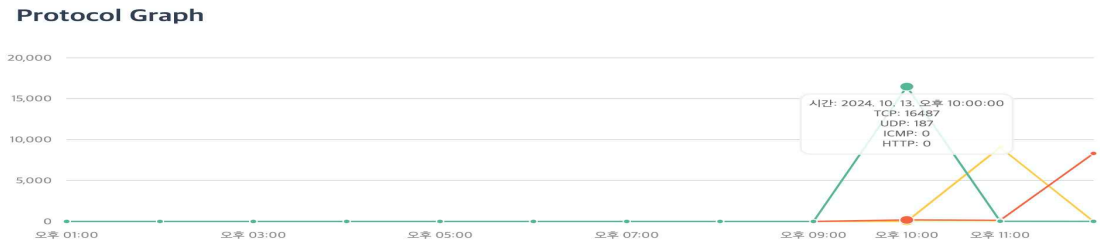
4.3.2 Protocal Ratio



[그림 33] 트래픽 프로토콜의 비율을 한눈에 보여주는 차트

위 그림은 같은 데이터를 사용하지만 다른 기준인 프로토콜을 분석한 차트로 몇 개의 프로토콜 패킷이 왔는지 쉽게 알 수 있게 해준다. 이는 자신의 서버가 어떤 프로토콜에 취약한지 분석하고 대응할 수 있다.

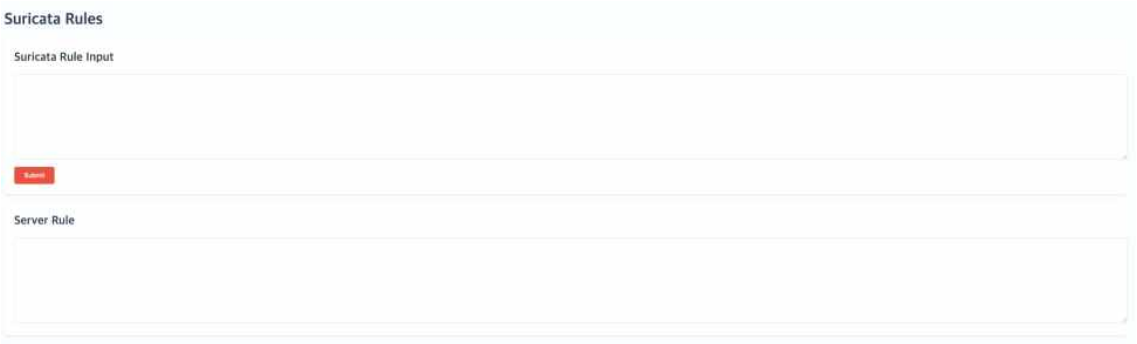
4.3.3 Protocol Graph



[그림 34] 트래픽 프로토콜이 온 시간을 보여주는 그래프

위 그림은 같은 데이터를 사용하지만, 또 다른 관점으로 표현한 그래프이다. 어떤 프로토콜을 사용한 공격이 언제 왔는지를 보여주어, 공격당한 시점을 분석하여 대응할 수 있다.

4.3.4 Suricata Rule



[그림 35] 수리카타 룰을 적용하고 적용된 룰 확인

위 그림은 자신(사용자)의 서버의 수리카타 룰을 적용하거나 적용된 룰을 확인하는 페이지이다. 위의 그래프들을 통해 분석한 내용을 토대로 자신(사용자)의 서버의 알맞은 수리카타 룰을 적용할 수 있다.

5. 결론

5.1 결론

이 보고서에서 제시한 시스템 구성은 DDoS와 같은 네트워크 공격으로부터 서버를 보호하기 위한 효과적인 솔루션을 제공한다. 자동화된 데이터 수집 및 분석이 가능하며, 트래픽의 비정상적인 변화를 감지할 수 있는 이 시스템은 네트워크의 안정성을 보장하는 중요한 도구로서 작동한다. 이를 통해 기업은 잠재적인 보안 위협을 미리 탐지하고 차단하여 서비스 중단과 같은 피해를 최소화할 수 있을 것이다. 또한, 지속적으로 변화하는 보안 환경에 대응하여 Suricata의 규칙을 유연하게 수정할 수 있으므로 향후 발생할 수 있는 새로운 유형의 공격에도 빠르게 대처할 수 있다.

5.2 기대효과

기대효과로는, 첫째, 네트워크 트래픽 모니터링을 통해 잠재적인 네트워크 공격을 초기에 탐지하고 차단할 수 있어 서버 가동률이 향상된다. 둘째, 자동화된 경고 시스템을 통해 관리자가 신속하게 문제에 대응할 수 있어 보안 관리의 효율성이 높아진다. 셋째, 대규모 트래픽 데이터를 효과적으로 처리하는 Kafka와 MongoDB의 활용은 시스템의 확장성을 보장하며, 기업의 데이터 처리 및 보안 역량을 증대시킨다. 마지막으로, ELK 스택을 통한 시각화는 직관적인 데이터 분석을 가능하게 하여 관리자들이 명확하고 빠르게 보안 상황을 파악할 수 있도록 돕는다. 이러한 시스템은 궁극적으로 기업의 IT 인프라 보안을 한층 강화하고, 지속적으로 변화하는 사이버 위협에 대비할 수 있는 능력을 제공할 것이다.

5.3 추후 보완사항

AI 기반의 이상 탐지 시스템 도입: AI 기반 시스템은 기존에 정의된 규칙 외에도 패턴화되지 않은 새로운 공격 유형을 학습하고 예측할 수 있기 때문에 더욱 정교한 방어 체계를 구축할 수 있다. 이를 통해, 알려지지 않은 제로데이 공격이나 비정상적인 트래픽 패턴을 조기에 탐지할 수 있으며, 관리자에게 보다 신뢰성 높은 경고를 제공할 수 있을 것이다. 또한, AI 모델의 학습 데이터를 주기적으로 갱신함으로써 시스템의 탐지 성능을 지속적으로 개선할 수 있다.

6. 별 첨

6.1 팀원 소개

√ 이두리 (팀장)
역할 : 백엔드 개발
개인 깃허브 주소 : https://github.com/DoolyDoori

√ 박주형 (팀원)
역할 : 백엔드 개발
개인 깃허브 주소 : https://github.com/pppppark

√ 송경선 (팀원)
역할 : 백엔드 개발
개인 깃허브 주소 : https://github.com/songkungsun

√ 김동철 (팀원)
역할 : 백엔드 개발
개인 깃허브 주소 : https://github.com/kdc3246

√ 이라규 (팀원)
역할 : 프론트엔드 개발
개인 깃허브 주소 : https://github.com/ragyu

√ 이건우 (팀원)
역할 : 프론트엔드 개발
개인 깃허브 주소 : https://github.com/lgw7537

6.2 발표 자료

(별도 첨부)