KIPS Trans. Comp. and Comm. Sys.
Vol.8, No.12 pp.287~296 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2019.8.12.287

Evaluation of Distributed Intrusion Detection System Based on MongoDB

HyoJoon Han† · HyukHo Kim†† · Yangwoo Kim†††

ABSTRACT

Due to the development and increased usage of Internet services such as IoT and cloud computing, a large number of packets are being generated on the Internet. In order to create a safe Internet environment, malicious data that may exist among these packets must be processed and detected quickly. In this paper, we apply MongoDB, which is specialized for unstructured data analysis and big data processing, to intrusion detection system for rapid processing of big data security events. In addition, building the intrusion detection system(IDS) using some of the private cloud resources which is the target of protection, elastic and dynamic reconfiguration of the IDS is made possible as the number of security events increase or decrease. In order to evaluate the performance of MongoDB - based IDS proposed in this paper, we constructed prototype systems of IDS based on MongoDB as well as existing relational database, and compared their performance. Moreover, the number of virtual machine has been increased to find out the performance change as the IDS is distributed. As a result, it is shown that the performance is improved as the number of virtual machine is increased to make IDS distributed in MongoDB environment but keeping the overall system performance unchanged. The security event input rate based on distributed MongoDB was faster as much as 60%, and distributed MongoDB-based intrusion detection rate was faster up to 100% comparing to the IDS based on relational database.

Keywords : Big Data, Intrusion Dectection System, MongoDB, Cloud Computing, Distributed Processing

MongoDB 기반의 분산 침입탐지시스템 성능 평가

한 효 준† · 김 혁 호†† · 김 양 우†††

요 약

IoT, 클라우드 컴퓨팅과 같은 인터넷 서비스의 발전과 사용량의 증가로 인해 수많은 패킷들이 인터넷상에서 빠르게 생성되고 있다. 안전한 인터넷 사용 환경을 만들기 위해서는 이 수많은 패킷 중에 존재할 수 있는 악성 데이터의 빠른 처리가 이뤄져야 한다. 본 논문에서는 빅데이터 보안 이벤트의 신속한 처리를 위해 비정형 데이터 분석과 빅데이터 처리에 특화된 MongoDB를 침입탐지시스템에 적용하였다. 또한 보호 대상인 사설 클라우드의 일부 자원을 이용하여 침입탐지시스템을 구축함으로써 증가 또는 감소하는 보안 이벤트 수에 따라 탄력적으로 컴퓨팅 자원 재구성이 가능하도록 하였다. 본 논문에서 제안하는 MongoDB 기반 침입탐지시스템의 성능을 평가하기 위하여 MongoDB 기반의 침입탐지시스템과 기존의 관계형 데이터베이스를 기반으로 한 침입탐지시스템의 프로토타입을 구축하고 성능을 비교하였다. 또한 분산화 구성에 따른 성능 변화를 확인하기 위하여 가상머신의 수를 변경하며 성능 변화를 확인하였다. 그 결과 전체적으로 MongoDB 환경에서 동일한 성능의 시스템을 분산화시켜 가상 머신의 수를 증가시킬수록 침입탐지시스템의 성능이 향상되는 것을 확인하였다. 분산 MongoDB 기반의 보안 이벤트 저장 속도가 관계형 데이터베이스 기반에 비해 최대 60%, 그리고 분산 MongoDB 기반의 침입 데이터 탐지 속도가 관계형 데이터베이스 기반에 비해 최대 100% 빠른 결과를 얻었다.

키워드 : 빅데이터, 침입탐지시스템, MongoDB, 클라우드 컴퓨팅, 분산 처리

1. 서 론

클라우드 컴퓨팅, IoT(Internet of Things)와 같은 인터넷 기반의 서비스들이 급속도로 발전하였다. 이와 더불어 이들의 사용량 또한 꾸준히 증가하며 많은 데이터를 생성하고 있다. 이러한 인터넷 기반 서비스들은 IoT 센서에서 발생한 데이터를 가공하기 위하여 서버와 데이터를 주고받거나 로컬 작업들을 온라인 서버로 가져와서 처리를 한다[1]. 그러나 이 경우 인터넷 패킷이 증가함에 따라 보안 이벤트의 개수가 너무 많아져서 보안이벤트가 빅데이터화 된다. 기존의 침입탐지시스템은 관계형 데이터베이스를 기반으로 만들어졌는데 관계형 데이터베이스는 하드웨어의 물리적, 비용적인 한계와

※본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터 지원 사업의 연구결과로 수행되었음(IITP-2019-2016-0-00465).
† 준 회 원 : 동국대학교 정보통신공학과 박사과정
†† 비 회 원 : 삼성전자 종합기술원 전문연구원
††† 종신회원 : 동국대학교 정보통신공학과 교수
Manuscript Received : June 17, 2019
First Revision : October 1, 2019
Accepted : October 24, 2019
* Corresponding Author : Yangwoo Kim(ywkim@dongguk.edu)

데이터 정형화의 효율성으로 인하여 빅데이터를 처리하기가 어렵다[2]. 따라서 기존의 관계형 데이터베이스 기반 침입탐지시스템으로 빅데이터 보안 이벤트를 빠르게 처리하고 관리하기가 어려워졌다.

서버를 오고가는 대부분의 정상적인 패킷 가운데 간혹 존재할 수 있는 악성 데이터를 탐지하는 것은 매우 중요하다. 악성 데이터가 서버에 침입하였을 때, 악성데이터를 빠르게 감지하고 처리하지 못한다면 고객 데이터와 기밀 데이터 등의 주요 데이터에 대한 유출이나 심각한 손상 등이 발생할 수 있다. 이러한 피해는 데이터에 그치지 않고 금전적인 피해도 발생할 수 있기에 세심한 관리가 필요하다.

많은 기업들은 관계형 데이터베이스를 기반으로 만들어진 침입탐지시스템을 도입하여 악성 데이터를 탐지하고 처리한다. MySQL 또는 MariaDB 같은 관계형 데이터베이스는 데이터간의 관계를 형성하고 저장하고 관리하는 특성을 가지고 있다[3]. 관계형 데이터베이스[2, 4]는 데이터의 ACID(Atomicity, Consistency, Isolation, Durability)를 보장하고 정규화와 인덱스 같은 기능을 통해 높은 성능을 보이는 장점을 가지고 있다. 하지만 스키마 구조로 인해 노드의 분산 확장이 어렵기 때문에 수평적 확장이 힘들 뿐만 아니라 고성능 장비를 구입하기 위해서는 막대한 비용이 들기 때문에 수직적 확장에도 어려움이 따른다. 따라서 관계형 데이터베이스를 사용한 침입탐지시스템으로는 빅데이터 보안 이벤트를 저장하고 처리하기 어렵다. 또한, 비정형 데이터를 처리하는 경우에는 데이터를 테이블 형식에 맞게 변형하는 전처리 과정을 거치기 때문에 성능저하가 발생한다. 본 논문에서는 위 문제점을 해결하고자 비정형데이터 분석과 빅데이터 처리에 특화된 NoSQL을 침입탐지시스템에 적용하였다. NoSQL은 수평적 확장을 통한 분산처리가 가능하고 비정형 데이터를 많은 가공 없이 처리할 수 있기 때문에 빅데이터화된 보안 이벤트들을 경제적이면서 신속하게 처리할 수 있다. 다양한 데이터 모델을 가진 NoSQL 중에서 빅데이터 보안 이벤트를 저장하고 처리하기 적합한 모델을 찾기 위해 HBase, Cassandra, MongoDB를 이용한 보안 이벤트 처리 시스템을 구축하여 실험을 진행하였다. 위 모델 중 MongoDB를 이용한 보안 이벤트 처리 시스템의 성능이 가장 우수하였기 때문에[5] 빅데이터 보안 이벤트를 신속하고 간편하게 처리하기 위하여 MongoDB를 사용하여 침입탐지시스템을 구축하였다.

관계형 데이터베이스는 데이터베이스 단에서 수평적 확장을 지원하지 않지만 미들웨어 단에서의 샤딩 기능을 제공하고 있다. 관계형 데이터베이스에서 샤딩 기능을 사용하기 위해 다양한 미들웨어 도구가 존재하는데 본 논문에서는 MariaDB[6] 데이터베이스에서 사용 가능한 Spider Storage Engine[7]를 사용하여 관계형 데이터베이스의 분산 환경 실험을 진행한다.

침입탐지시스템에서 처리해야 하는 보안 이벤트의 양은 수시로 변하고 그 때문에 예측하기 어렵다. 따라서 침입탐지시스템을 구축할 경우에 적정 성능의 시스템 규격을 결정하는데 어려움을 겪을 수 있다. 저사양의 시스템을 구성할 경우에는 갑자기 많은 패킷이 유입되었을 때 신속하게 처리하기 어렵다. 반대로 고사양의 시스템을 구성할 경우에는 적은 양의 패킷이 들어오는 동안 많은 자원이 낭비되는 비효율성 문제를 갖는다. 이러한 문제들을 고려하여 본 논문에서는 신속하고 능동적으로 컴퓨팅 자원의 투입과 회수가 자동으로 이루어지는 클라우드 컴퓨팅 환경에서 침입탐지시스템을 구축하였다.

클라우드 컴퓨팅 자원은 간단하게 대여하고 반납할 수 있기 때문에 데이터가 탄력적으로 변하는 상황에서 유용하게 사용할 수 있다[8]. 보안 이벤트는 시간과 상황에 따라 유입되는 양에 많이 차이가 나기 때문에 클라우드 컴퓨팅 자원을 사용하면 자원의 낭비를 줄이며 신속하게 보안 이벤트를 처리할 수 있다. 하지만 전체적으로 동일한 성능의 침입탐지시스템 서버를 이용하여도 데이터베이스 서버의 구성에 따라 침입탐지시스템의 성능이 변할 수 있기 때문에 데이터베이스 서버 구성에 따른 침입탐지시스템의 성능 변화를 확인할 필요가 있다.

본 논문에서는 제안한 MongoDB 기반 침입탐지시스템의 우수성을 확인하기 위하여 MongoDB 기반 침입탐지시스템과 관계형 데이터베이스 기반 침입탐지시스템의 프로토타입 시스템을 구축하여 보안 이벤트 처리 성능을 비교 평가하였다. 이와 더불어 데이터베이스 서버의 개수에 따른 성능 변화를 관찰하기 위하여 데이터베이스 개수를 변동시키며 분산 MariaDB와 분산 MongoDB 침입탐지 시스템을 구축하였고, 이를 통해 분산 MongoDB 데이터베이스 환경에서 동일한 성능 대비 데이터베이스 서버가 증가함에 따라서 침입탐지시스템의 성능이 증가함을 확인하였다.

본 논문은 다음과 같이 구성된다. 우선, 관련 연구인 2장에서는 MongoDB와 침입탐지시스템, 클라우드 컴퓨팅, 그리고 Spider Storage Engine에 대해 살펴본다. 3장에서는 MongoDB기반의 침입탐지시스템을 설계하고 프로토타입을 구축한다. 4장에서는 구축한 프로토타입 시스템의 성능을 비교 평가한다. 마지막으로 결론인 5장에서는 성능 결과를 분석한다.

2. 관련 연구

2.1 MongoDB

MongoDB[9, 10]는 신뢰성과 확장성에 기반한 문서 지향 데이터베이스로, 빅데이터 환경에서의 사용 편의성과 낮은 관리 비용을 목표로 개발되었다. MongoDB는 저장의 최소 단위로 관계형 데이터베이스의 행과 유사한 문서를 사용한다. 각 문서들은 데이터베이스가 관리하는 컬렉션이라는 단위로 저장된다. MongoDB는 분산 데이터베이스 구성을 위해 자동 샤딩(Auto-Sharding)을 사용한다. 샤딩은 여러 데

이터베이스 서버에 데이터를 나누어 저장하는 과정을 말하며, 데이터를 여러 서버로 분할 저장하기 때문에 더 많은 데이터를 관리하고 처리할 수 있다. MongoDB는 자동 샤딩을 기본적으로 제공하기에 응용 프로그램 단에서 어려움 없이 데이터를 분산 처리할 수 있다. 자동 샤딩 기능은 데이터의 양이 증가하거나 서버의 용량이 부족할 때 사용할 수 있다. 데이터의 양에 상관없이 시스템 속도를 안정적으로 작동시킬 수 있으며 용량 부족으로 인하여 데이터를 지우거나 분산 처리를 위해 데이터베이스의 구조를 변경할 필요가 없다.

2.2 침입탐지시스템

침입탐지시스템[11]은 네트워크를 통해 오고가는 패킷들의 감시를 통하여 침입 발생여부를 탐지하고 대응하는 시스템이다. 로컬 네트워크나 호스트에 위치하여 컴퓨터 사용 자원의 가용성, 무결성, 기밀성을 저해하는 행위를 탐지하고 네트워크 공격이나 시스템에 대한 불법 접근 및 행위 등에 대해 대응방안을 세울 수 있게 해준다. 침입탐지시스템은 일반적으로 네 단계의 기술적인 구성을 가진다. 첫 번째 단계는 침입탐지를 위한 데이터를 수집하는 정보수집 단계이다. 네트워크를 오고가는 패킷을 캡처하거나 호스트에 기록된 데이터를 저장한다. 두 번째 단계는 수집한 정보를 쉽게 판단하기 위하여 알맞은 형태로 데이터를 가공하는 정보가공 단계이다. 세 번째 단계는 정보 분석 및 탐지 단계이다. 가장 중요한 단계로써 가공된 정보를 분석하여서 데이터의 침입 여부를 판정한다. 네 번째 단계는 분석된 침입 데이터 결과를 관리자에게 보고하는 보고 단계이다. 침입탐지시스템은 위의 네 가지 단계를 거쳐서 보안 이벤트를 처리한다.

침입탐지시스템은 탐지방법에 따라 오용 기반 탐지와 비정상 행위 탐지로 분류할 수 있다.

오용 탐지 방식은 이미 분석이 완료된 공격으로부터 특정 시그니처(Signature)를 추출하여 저장한다. 그 이후 분석이 필요한 패킷의 시그니처와 공격 시그니처를 비교하여 침입 여부를 판단하는 방식이다. 트로이 목마, 백도어 공격 등을 빠르게 탐지할 수 있으며 오탐률이 낮다. 하지만 시그니처 룰에는 분석이 완료된 공격만 기록되기 때문에 룰에 적혀 있지 않은 새로운 공격은 탐지할 수 없다. 따라서 지속적으로 시그니처 룰을 업데이트할 필요가 있다. 또한, 대용량의 이벤트를 한 번에 처리하거나 시그니처 룰이 많아질 경우 모든 데이터를 일대일로 패턴 매칭 해야 하기 때문에 속도저하가 발생할 수 있다.

비정상 행위 탐지 방식은 네트워크의 정상적인 사용상황을 정량적, 통계적으로 분석하여 정상 행위의 범주를 지정한다. 그리고 지정된 정상 행위의 범주를 벗어나는 모든 행위를 비정상 행위로 간주하고 침입으로 판단한다. 비정상 행위 탐지 방식은 인공지능 알고리즘을 사용한 학습을 통해서 정상 행위를 스스로 판단하기 때문에 정상 행위의 범주를 수작업으로 조정할 필요가 없다. 또한 스스로 학습하기 때문에 새로운 유형의 공격이 들어와도 탐지할 수 있다. 하지만 정상 행위 범주를 설정하기 어렵고 오탐률이 높다는 단점이 있다.

침입탐지시스템은 데이터를 수집하는 주체에 따라 네트워크 기반 침입탐지시스템과 호스트 기반 침입탐지시스템으로 구분된다. 네트워크 기반 침입탐지시스템은 네트워크 단에서 설치되어 네트워크의 트래픽을 모니터링하고 분석한다. 네트워크 단에서 실행되기 때문에 서버의 성능 저하 없이 네트워크에서 발생하는 여러 유형의 네트워크 공격을 탐지할 수 있지만 트래픽의 양이 많아질수록 성능이 저하된다. 호스트 기반 침입탐지시스템은 개인의 컴퓨터 또는 서버에 직접 설치된다. 호스트 성능에 의존적이며 시스템 내부의 데이터를 감시하고 분석한다.

대표적인 오픈소스 오용 탐지 침입탐지시스템으로 스노트(Snort)가 있다[12]. 1998년 처음 공개된 스노트는 단순한 패킷 스니핑(Packet Sniffing) 프로그램이었다. 1999년에 룰 기반의 분석 기능이 스노트에 추가된 이후 개발자 커뮤니티를 통해 지속적으로 기능이 보완되었다. 결정적으로 'Sourcefire'에 인수되면서 오용 탐지 침입탐지시스템의 사실상 표준 기술이 되었다. 스노트의 기능은 패킷 스니핑, 패킷 로깅(Packet Logging) 그리고 네트워크 침입탐지시스템으로 구분할 수 있다. 패킷 스니핑은 네트워크의 패킷을 읽어서 보여주는 기능이다. 패킷 로깅은 스니핑한 패킷을 저장하고 로그에 남기는 기능이다. 네트워크 침입탐지시스템 기능은 네트워크 트래픽을 분석하여 공격을 탐지하는 기능으로 버퍼 오버플로우, 포트 스캔 등의 공격을 탐지할 수 있다. 스노트는 4가지 구성요소로 이루어지는데 스니핑 기능을 제외한 기능들은 플러그인(Plug-in) 형태로 제공된다.

침입탐지시스템의 성능을 향상시키기 위하여 다양한 형태의 인공지능 알고리즘을 시스템에 적용하는 연구가 활발히 이루어지고 있다[13, 14]. 이와 같은 연구들은 침입탐지시스템이 인공지능 기술을 활용하여 새로운 유형의 공격을 스스로 찾을 수 있게 함에 있다. 본 논문은 시스템의 자동화에 초점을 두지 않고 보안 이벤트를 저장하고 관리하는 데이터베이스 시스템의 변화에 따른 성능 증감을 관찰하는 것을 목표로 하고 있다.

본 논문에서 구축하는 침입탐지시스템은 오용탐지 방식의 네트워크 침입탐지시스템으로 스노트 시그니처 룰을 사용하여 공격을 탐지한다.

2.3 클라우드 컴퓨팅

클라우드 컴퓨팅은 서버, 데이터베이스, 소프트웨어 등의 컴퓨팅 서비스를 인터넷을 통해 접근하여 제공받는 컴퓨팅 패러다임이다[15, 16]. 클라우드 서비스 제공자로부터 필요한 자원만 빌려서 사용하고 사용한 만큼만 비용을 지불한다. 제공되는 서비스의 유형에 따라 서버, 저장소, 네트워크 등의 인프라를 제공하는 IaaS(Infrastructure as a Server), IaaS

자원을 포함한 운영체제와 개발 도구 등의 플랫폼 서비스를 제공하는 PaaS(Platform as a Service), 클라우드 환경에서 동작하는 응용프로그램을 제공하는 SaaS(Software as a Service)로 구분된다.

본 논문에서는 사설 클라우드 환경에 침입탐지시스템을 구축하여 사용하는데, 사설 클라우드 환경을 구축하기 위하여 IaaS 형태의 오픈소스 프로젝트인 오픈스택[17](OpenStack)을 사용한다.

오픈스택은 Rackspace사와 미국의 NASA가 설립한 오픈소스 프로젝트이다. 오픈스택은 컴퓨트, 오브젝트 스토리지, 이미지, 인증 서비스 등을 프로젝트 단위로 나누어 구성하고 동작하도록 하고, 프로젝트 간에 REST API를 사용하여 프로젝트들이 유기적으로 연결되어 하나의 커다란 클라우드 컴퓨팅 시스템으로 구축되게 만들었다.

2.4 Spider Storage Engine

Spider Storage Engine[7]은 MariaDB에서 개발된 샤딩 기능이 내장된 저장 엔진이다. Spider Storage Engine은 NoSQL의 샤딩 기능과는 달리 데이터베이스 단에서 동작하는 것이 아니라 미들웨어 단에서 동작한다. 서로 다른 MariaDB 인스턴스의 테이블을 미들웨어 단에서 하나의 인스턴스처럼 처리할 수 있다. Spider Storage Engine을 사용하여 테이블을 만들면 원격 서버의 테이블과 링크를 만들게 되고 이 링크를 통해 샤딩 기능을 구현할 수 있게 된다.

3. MongoDB 기반의 침입탐지시스템

본 논문에서 설계하고 구축하는 MongoDB 기반의 침입탐지시스템은 IaaS 형태의 클라우드 컴퓨팅 시스템 내부에 설치되어 동작한다.

3.1 MongoDB 기반 침입탐지시스템 설계

본 논문에서 설계한 MongoDB기반의 침입탐지시스템의 구성은 다음 Fig. 1과 같다.

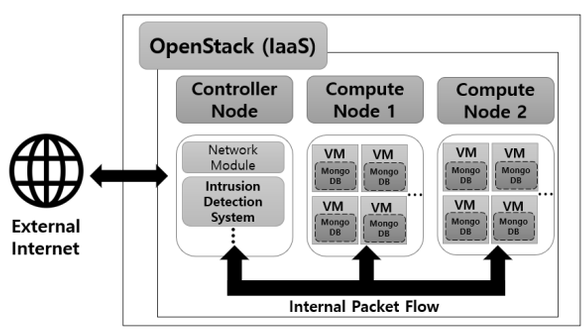

Fig. 1. Intrusion Detection System Configuration in Cloud Computing Environment

침입탐지시스템과 데이터베이스 서버가 설치되는 환경은 오픈스택을 이용하여 구성하였다. 침입탐지시스템은 오픈스택 내부 노드를 관리하는 컨트롤러 노드(Controller Node)에 설치를 하였고, 데이터베이스 서버는 컴퓨트 노드(Compute Node) 내부에 생성되는 가상머신(Virtual Machine, VM)의 일부분을 사용하였다. 컴퓨트 노드의 개수는 사용해야할 가상머신의 개수에 따라 자유롭게 확장이 가능하다. 가상머신의 생성요청이 발생하면 컨트롤러 노드는 컴퓨트 노드의 현재 상태에 따라 알맞은 위치에 가상머신을 생성한다. 침입탐지시스템을 위한 데이터베이스는 임의의 가상머신에 설치되며 실험 환경에 따라서 한 대의 가상머신부터 여덟 개의 가상머신까지 사용될 수 있다. 또는 오픈 스택을 이용해 구성한 사설 클라우드 전체의 규격에 따라 알맞게 데이터베이스 서버의 양을 조절할 수 있다. 침입탐지시스템은 Fig. 2와 같이 패킷 스니핑(Packet Sniffing), 전처리기(Pre-Processor), 보안 이벤트 로그 및 저장(Secure Event Log & Store), 침입 탐지(Intrusion Detection)로 이루어져 있다. 패킷 스니핑은 네트워크 트래픽을 캡처하여 패킷을 살펴보는 역할을 한다. 전처리기에서는 패킷을 데이터베이스에 저장하기 전에 패킷을 가공하여 보안 이벤트로 변환한다. 보안 이벤트 로그 및 저장에서는 전처리기에서 가공한 보안 이벤트를 데이터베이스에 저장하는 역할을 한다. 마지막으로 침입 탐지는 시그니처 룰을 이용해 데이터베이스에 저장된 악성 데이터를 탐지하는 역할을 한다.

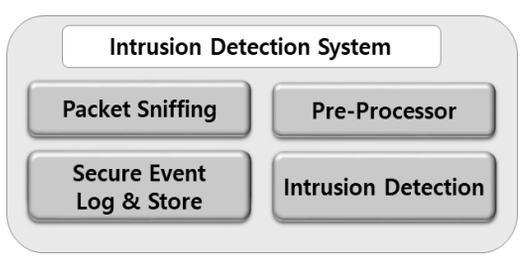

Fig. 2. Intrusion Detection System Components

데이터베이스 서버는 침입탐지시스템에서 생성된 보안 이벤트를 저장하고 침입탐지 요청이 오면 시그니처 룰에 맞추어 패턴 매칭을 실행하여 결과를 출력한다. 본 논문에서는 MongoDB를 데이터베이스로 사용하였고 신속한 빅데이터 처리를 위해 분산처리 환경으로 구성하였다.

침입탐지시스템이 실행되는 프로세스는 Fig. 3과 같다. 패킷 스니핑을 통해 캡처한 패킷을 전처리기에서 가공을 한 후 보안 이벤트를 데이터베이스에 저장한다. 이후 저장된 보안 이벤트를 통해 침입탐지를 하게 되고, 침입데이터가 발견되면 알람을 통해 관리자에게 알리게 된다. 데이터베이스에 저장된 보안 이벤트는 추후에 관리자가 보안 이벤트를 분석할 때 사용한다. 침입탐지를 살펴보면 탐지 룰을 설정하는 부분과 패턴 매칭 부분, 그리고 침입데이터 추출 부분으로 나뉜

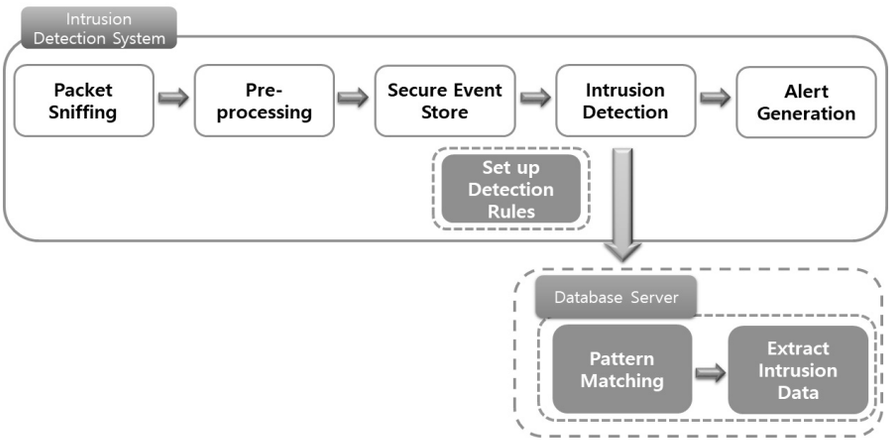

Fig. 3. Intrusion Detection System Execution Process

다. 탐지 룰 설정은 침입탐지시스템에서 이루어지는데 탐지 룰은 데이터베이스에 저장되어 있으며 관리자가 수동으로 룰을 추가할 수 있다.

본 논문에서 설계하는 침입탐지시스템의 관제 대상은 클라우드 서버 내부로 접근하는 모든 인터넷 트래픽이다. 컨트롤러 노드의 네트워크 모듈이 클라우드 서버 내 가상머신들의 라우터 역할을 하여 외부 인터넷과 통신하기 때문에 침입탐지시스템을 컨트롤러 노드에 설치함으로써 클라우드 서버 전체를 대상으로 침입을 탐지할 수 있다. 또한, 침입탐지시스템은 IaaS 환경의 컨트롤러 노드뿐만 아니라 일반 서버의 네트워크 관리 노드 혹은 라우터 등 네트워크를 모니터링할 수 있는 다양한 시스템에 설치될 수 있다.

3.2 MongoDB 기반 침입탐지시스템 구축

본 논문에서 제안한 설계도에 따라 오픈스택 환경에 침입탐지시스템을 구축하였다. 오픈스택 환경은 컨트롤러 노드 1대, 컴퓨트 노드 2대로 구성되어 있으며 각 노드에 대한 사양은 다음 Table 1과 같다.

Table 1. OpenStack Configuration Node Specification

	Controller Node	Compute Node
CPU	AMD A10 7870K (4 Core)	AMD FX 7870 (8 Core)
RAM	DDR3 4GB	DDR3 8GB

컨트롤러 노드는 관리용 노드이기 때문에 상대적으로 저사양이고, 컴퓨트 노드는 실제 가상머신이 생성되고 작동하는 노드로 컨트롤러 노드보다 상대적으로 고사양이다. 컨트롤러 노드에는 오픈스택을 운영하기 위한 소프트웨어들과 본 논문에서 제안한 침입탐지시스템이 구축되었다. 컴퓨트 노드에는 오픈스택을 운영하기 위한 소프트웨어들과 관리자와 사용자에 의해 생성된 가상머신들이 존재한다. 본 논문에서는 컴퓨트 노드에 생성된 가상머신의 일부에 데이터베이스를 설치하여 보안 이벤트 관리와 처리를 위한 데이터베이스 서버로 사용하였다. 침입탐지시스템의 구축은 패킷을 캡처하여 데이터베이스에 저장하는 보안 이벤트 저장 단계와, 저장된 보안 이벤트로부터 침입 데이터를 탐지하는 침입탐지 단계로 구분한다.

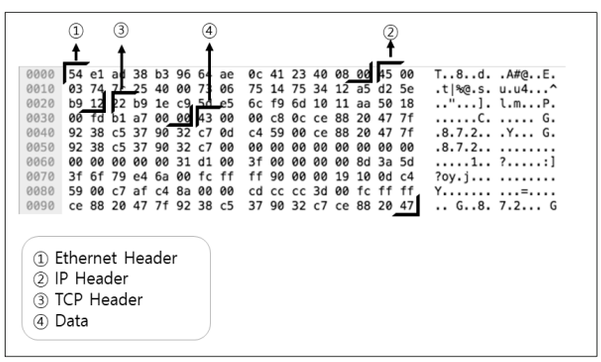

Fig. 4. Packet Structure

1) 보안 이벤트 저장 단계 구축

보안 이벤트 저장 단계는 패킷 스니핑을 통해 패킷을 캡처하고 전처리 단계를 거쳐 데이터베이스에 저장하는 순서로 이루어진다. 본 논문에서는 실시간 네트워크 트래픽 대신 미리 캡처된 pcap(packet capture) 파일을 사용하여 비교 실험 환경의 동일성을 보장해준다. 다음 Fig. 4는 패킷의 구조이다. 패킷은 길이에 따라서 이더넷 헤더, IP 헤더, UDP/TCP 헤더와 데이터 영역으로 구분이 된다.

패킷의 구조를 정리하면 Table 2와 같은 결과를 얻을 수 있다. TCP와 UDP 말고도 다양한 종류의 프로토콜이 존재하지만 본 논문에서는 실험의 간편성을 위해서 데이터 전송을 위해 주로 사용되는 TCP와 UDP 프로토콜을 사용한 패킷만 전처리 과정을 거쳐서 데이터베이스에 저장하였다.

보안 이벤트 저장은 패킷 데이터 세트인 pcap 파일을 읽고, 한 패킷씩 불러와 전처리 과정을 거친다. 전처리 과정에

서는 패킷의 IP와 프로토콜의 종류를 확인한 후 종류에 맞추어 패킷을 분할하고 배열에 저장한다. 배열에는 이더넷 헤더, IP 헤더, UDP/TCP 헤더, 그리고 데이터로 나누어 저장 한다. 최종적으로 데이터베이스에 나눠진 데이터를 저장하여 보안 이벤트 저장 단계를 끝낸다.

Table 2. Length by Packet Type

Packet Type	Packet Data Length
Ethernet Header	28 Hex
IPv4	40 Hex
IPv6	80 Hex
TCP	40 Hex
UDP	16 Hex

2) 침입데이터 탐지 단계 구축

침입데이터 탐지 단계는 앞서 보안 이벤트 저장 단계를 통해 데이터베이스에 저장된 보안 이벤트들을 대상으로 패턴 매칭을 통해 침입데이터를 탐지하는 단계이다.

침입탐지시스템에는 분석된 공격 시그니처가 저장된 탐지 룰이 설정되어 있다.

탐지 룰은 직접 설정할 수 있으며 일반적으로 데이터베이스에 저장된 탐지 룰을 사용한다. 탐지 룰에 의해서 데이터베이스의 검색 조건을 설정하는데, 패턴 매칭을 위해 부분 단어 검색이 가능한 검색 조건을 설정해야 한다. 이후에 설정한 검색 조건을 이용하여 보안 이벤트가 저장된 데이터베이스에서 침입데이터를 찾아 탐지하게 된다.

4. 침입탐지시스템 성능 평가 실험

4.1 실험 환경

본 연구에서 실험하는 침입탐지시스템은 오픈스택으로 구성된 사설 클라우드 내부에 구축되었다. Fig. 1과 같이 컨트롤러 노드 내부에 침입탐지시스템이 설치되어있다. 컨트롤러 노드의 사양은 Table 1과 같다. 보안 이벤트가 저장되는 데이터베이스는 컴퓨트 노드 내부에 생성되는 가상머신을 이용한다. 실험은 가상머신이 한 대인 분산되지 않은 환경에서의 데이터베이스서버와 가상머신이 두 대 이상인 분산 환경에서의 데이터베이스 서버를 사용해 구성하였다. 가상머신의 수가 달라져도 가상머신 성능의 총 합은 동일하다. Table 3은 실험에서 사용한 다양한 가상머신 구성들을 나타낸다.

환경 1R(Environment 1R)의 데이터베이스 서버에는 가상머신 1대에 관계형데이터베이스인 MariaDB가 설치된다. 환경 2R(Environment 2R), 환경 3R(Environment 3R), 환경 4R(Environment 4R) 에서는 가상머신이 2대 이상일 때 MariaDB와 Spider Storage Engine을 사용하여 분산처리 환경을 구축하였다. 환경 1N(Environment 1N)은 환경 1R과 CPU, RAM 등의 성능이 모두 동일 하지만 MariaDB 데이터베이스 대신 본 연구에서 제안하는 MongoDB를 설치하였다. 환경 2N(Environment 2N), 환경 3N(Environment 3N), 환경 4N(Environment 4N)에서는 가상머신이 2대 이상일 때 MongoDB 데이터베이스를 사용하여 분산처리 환경으로 구축하였다. 환경 1R부터 환경 4N 까지 모든 실험 환경의 총 성능은 동일하게 구성되었다. 데이터베이스는 사설 클라우드 서버 안에 존재하는 임의의 가상머신에 설치되어 사용된다. 가상머신은 사용하고자 하는 성능에 따라 컨트롤러 노드에 의해 설치되는 컴퓨트 노드의 위치가 결정되며 컴퓨트 노드의 개수나 가상머신이 설치되어 있는 컴퓨트 노드의 위치는 침입탐지시스템의 성능과는 무관하다.

Table 3. Various Database Server Configurations

Environment	CPU	RAM	Number of VM	Database
Environment 1R	8 Core	16 GB	1	MariaDB
Environment 2R	4 Core	8 GB	2	MariaDB
Environment 3R	2 Core	4 GB	4	MariaDB
Environmen 4R	1 Core	2 GB	8	MariaDB
Environment 1N	8 Core	16 GB	1	MongoDB
Environment 2N	4 Core	8 GB	2	MongoDB
Environment 3N	2 Core	4 GB	4	MongoDB
Environment 4N	1 Core	2 GB	8	MongoDB

실험에 사용한 프로그램, 데이터베이스 그리고 라이브러리에 대한 정보는 Table 4와 같다.

Table 4. Usage Program

Type	Name	Version
Language	Python	2.7.12
Database	MongoDB	3.2.14
Database	MariaDB	10.3.18
Library	PyMongo	3.5.1
Library	PyMySQL	0.7.2
Engine	Spider Storage Engine	10.3.18

4.2 실험 데이터

본 논문에서 구축한 시스템을 평가하기 위해 진행하는 실험에서 사용하는 데이터는 실시간 네트워크 트래픽을 캡처하여 사용하지 않고 미리 저장된 pcap 파일을 사용하였다. 따라서 각각의 데이터베이스에서 동일한 보안 이벤트를 저장하고 처리함으로써 신뢰성 있는 평가 실험이 진행되도록 하였다. 실험을 위해 사용된 pcap 파일은 침입 탐지를 위해 생성

된 벤치마크 데이터 세트이다[18]. 약 40 GB 정도의 네트워크 패킷으로 이루어져 있는 데이트 세트로 분할된 4개의 pcap 파일로 구성 되어 있다. 각 pcap 파일의 크기는 각각 4.4 GB, 4.2 GB, 7.2 GB, 24.5 GB 이다. 이들 pcap 파일에 저장된 패킷을 하나씩 읽어서 전처리 과정 후 데이터베이스에 저장 하고, 이들을 대상으로 패턴매칭을 거쳐 침입 데이터를 탐지하게 된다.

4.3 실험 결과

구축한 다양한 구성의 환경에서 실험 데이터를 대상으로 보안 이벤트 저장 단계와 침입 데이터 추출 단계를 이용하여 실험을 진행 하였다. 다음 Table 5는 이벤트 저장 시간을 측정한 결과이다.

보안 이벤트 저장 실험 결과를 확인해보면 환경 1R은 단일 노드에서 MariaDB를 사용한 환경이며 환경 2R, 환경 3R, 환경 4R은 Spider Storage Engine을 사용하여 MariaDB 분산 처리 환경을 구성한 경우이다. 환경 1N은 단일 노드에서 MongoDB를 사용했을 경우이다. 환경 2N, 환경 3N, 환경 4N은 샤딩 기능을 사용하여 MongoDB 분산 처리 환경을 구성한 경우이다. 각 보안 이벤트 저장에 사용된 패킷의 수는 1차 저장에서는 5,958,965개, 2차 저장에서는 5,744,788 개, 3차 저장에서는 15,589,699개, 4차 저장에서는 40,697,693개를 저장하였다. 총 저장된 패킷은 67,991,145개이다. 저장 시간을 보면 MariaDB의 경우 단일 노드에서 동작할 때보다 분산 처리 환경을 사용할 때 분산 노드 수가 증가할수록 저장 시간이 급격하게 늘어남을 볼 수 있다. 반면에 MongoDB의 경우 단일 노드에서 동작할 때보다 분산 처리 환경을 사용할 때 분산 노드 수가 증가할수록 저장 시간이 감소함을 확인하였다.

전체적으로 MariaDB 환경에서의 보안 이벤트 저장시간보다 MongoDB 환경에서의 보안 이벤트 저장시간이 많이 단축되었음을 확인하였으며 그 중 가상 머신 8대로 MongoDB 분산 데이터베이스를 사용한 환경 4N이 가장 빠른 속도로 보안

Table 5. Security Event Store Time (sec)

Security Event Store	1st Store	2nd Store	3rd Store	4th Store
Environment 1R	17,516	17,545	49,127	130,851
Environment 2R	41,472	41,296	110,277	295,008
Environment 3R	69,002	70,432	194,223	516,238
Environment 4R	231,648	227,025	625,274	1,664,313
Environment 1N	12,488	12,728	36,144	101,168
Environment 2N	11,355	11,253	34,426	97,581
Environment 3N	10,984	10,865	32,957	94,802
Environment 4N	10,579	10,410	30,521	91,548

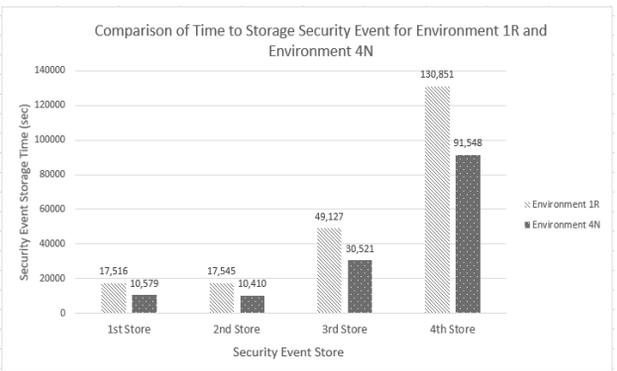

Fig. 5. Comparison of Time to Storage Security Event for Environment 1R and Environment 4N

이벤트를 저장하는 것을 확인하였다. 또한, 2차 저장에서는 패킷의 수가 1차 저장에서 저장된 패킷의 수보다 적음에도 불구하고 저장시간이 증가함을 보였는데, 이는 데이터베이스에 데이터를 저장할 때 기존에 존재하는 데이터가 있으면 데이터의 ACID를 보장하기 위한 추가 과정을 거치거나 인덱싱 과정을 거치는 등 기존에 저장되어 있는 데이터의 영향을 받음을 의미한다. 따라서 보안 이벤트가 증가하여 빅데이터화 될수록 영향을 받는 데이터가 증가하기 때문에 데이터를 저장하는 성능이 저하될 수 있음을 보았다.

Fig. 5는 MariaDB 환경 중 가장 우수한 성능을 보인 환경 1R과 MongoDB 환경 중 가장 우수한 성능을 보인 환경 4N의 보안 이벤트 저장 시간을 비교한 그래프이다. 1차 저장부터 4차 저장까지 환경 4N은 환경 1R보다 평균 60%의 시간 감소율을 보였다. MongoDB 환경을 사용하면 MariaDB를 사용할 때 보다 모두 빠른 속도로 보안 이벤트를 저장할 수 있었으며, 분산 노드의 수가 증가할수록 더욱 높은 성능을 나타냄을 확인하였다.

분산 MongoDB 데이터베이스가 일반적인 MongoDB 데이터베이스 환경 보다 빠르게 보안 이벤트를 저장할 수 있는 이유는 침입탐지시스템에 의해서 전처리 과정을 거친 보안 이벤트가 데이터베이스 서버에 저장되기 전에 버퍼에 머무르는데, 일반적인 MongoDB 데이터베이스는 버퍼에서 한 개의 보안 이벤트를 꺼내서 저장 하지만, 분산 MongoDB 데이터베이스는 다수의 보안 이벤트를 동시에 저장할 수 있고, 트랜잭션이 집중되지 않아 부하가 적기 때문에 분산되지 않은 환경 보다 빠른 속도로 저장을 할 수 있다. 또한, 분산 MongoDB 데이터베이스는 특별한 정형화 과정 없이 데이터를 저장할 수 있기 때문에 MariaDB 데이터베이스에 비해 보안 이벤트 저장 시간이 단축이 되다. MariaDB 데이터베이스는 스키마 구조가 엄격해서 데이터를 저장하기 이전에 정형화 작업을 통해 저장 가능한 형태로 데이터를 변형해야하기 때문에 정형화 과정에서 데이터 저장 시간이 증가한다. 그리고 동시 접근에 제한이 없는 MongoDB 데이터베이스에 비

해서 MariaDB 데이터베이스는 동시 접근에 제한이 심하고, 동시에 접근하는 데이터가 많을수록 데이터베이스의 성능이 저하되기 때문에 보안 이벤트 저장 시간이 MongoDB 데이터베이스에 비해 증가하는 것을 볼 수 있다. 또한 MariaDB 데이터베이스는 분산 노드가 증가할수록 보안 이벤트 저장 속도가 급격하게 증가하는 것을 볼 수 있는데, 이는 MariaDB가 Spider Storage Engine을 사용하여 데이터를 분산 저장할 때 데이터베이스의 오류를 대비하여 복제 데이터를 만들고 데이터들의 ACID를 보장하기 위한 추가 과정이 많아지기 때문이다. 또한 데이터베이스 단이 아닌 미들웨어 단에서 작업을 처리해야하기 때문에 ACID를 보장하기 위해 Spider Storage Engine과 데이터베이스와의 트랜잭션 부하로 인해 분산 노드의 개수가 증가할수록 많은 성능저하가 발생할 수 있다. 따라서 분산 노드가 2개 일 때는 단일 노드에 비해 보안 이벤트 저장 시간이 2배 늘어났지만 분산 노드가 8개 일 때는 단일 노드에 비해 보안 이벤트 저장 시간이 10배 이상 늘어나며 많은 성능 저하가 발생함을 확인하였다.

다음으로 저장된 보안 이벤트들을 대상으로 침입 데이터를 탐지하는 시간을 측정한 결과이다. 평균 침입 데이터 추출은 1차 저장, 2차 저장 등의 각 저장이 끝난 후 침입 데이터 탐지를 진행하였다. 또한 각 탐지 실험에서 5번씩 탐지 시간을 측정하여 평균값으로 침입 탐지 시간을 표현하였다. Table 6은 각 환경에서 침입 탐지 실험을 한 결과이다.

침입 데이터 탐지 실험 결과를 보면 다음과 같다. 1차 탐지에서는 저장된 5,958,965개의 패킷 중에서 침입 데이터 3,943개를 탐지하는데 걸리는 시간을 측정한 결과이다. 2차 탐지는 1차 저장과 2차 저장을 합한 11,703,753개의 패킷 중 4,332개, 3차 탐지는 1차 저장부터 3차 저장까지 합한 27,293,452개의 패킷 중 4,949개, 4차 탐지는 1차 저장부터 4차 저장까지 모든 저장이 끝난 후에 67,991,145개의 패킷 중 10,260개의 침입 데이터를 탐지한다. 탐지 시간을 보면 보안 이벤트 저장과 마찬가지로 MongoDB 환경이 MariaDB 환경보다 빠른 속도로 침입 데이터를 탐지하였으며 그 중 환경 4N에서 가장 빠른 속도로 침입 데이터를 탐지하는 것을 확인하였다.

Table 6. Intrusion Data Detection Time (sec)

Intrusion Data Detection	1st Detect	2nd Detect	3rd Detect	4th Detect
Environment 1R	94	183	397	948
Environment 2R	101	204	425	1,120
Environment 3R	165	322	658	1,582
Environment 4R	247	492	994	1,869
Environment 1N	40	150	240	629
Environment 2N	37	139	221	576
Environment 3N	35	129	208	547
Environment 4N	34	125	203	513

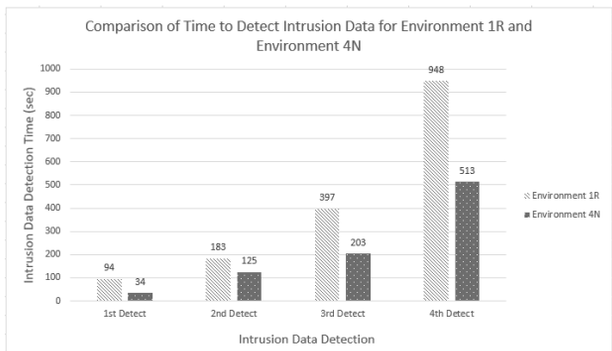

Fig. 6. Comparison of Time to Detect Intrusion Data for Environment 1R and Environment 4N

Fig. 6은 MariaDB 환경 중 가장 우수한 성능을 보인 환경 1R과 MongoDB 환경 중 가장 우수한 성능을 보인 환경 4N의 침입 데이터 탐지 시간을 비교한 그래프이다. 1차 탐지부터 4차 탐지까지 환경 4N은 환경 1R보다 평균 100%의 시간 감소율을 보이며 침입 데이터 탐지의 성능이 개선됨을 확인하였다. 또한 보안 이벤트 저장과 마찬 가지로 가상 머신의 수가 늘어남에 따라 침입 데이터 탐지 시간이 줄어드는 것을 확인할 수 있었으며, 추가적으로 분산 MongoDB 데이터베이스 환경에서 저장된 데이터의 양이 증가할수록 침입 데이터 탐지를 감지하는 시간의 감소 비율이 증가하는 것을 확인할 수 있었다. 분산 MongoDB는 빅데이터 환경에서 좋은 성능을 내기 때문에 실험에서 사용한 40GB 정도의 데이터가 아닌 실제 환경에서 수 백 GB, 또는 수 TB 이상의 빅데이터 보안 이벤트가 저장 될 시 관계형 데이터베이스에 비해 더욱 성능 폭의 차이가 증가할 것으로 기대된다.

분산 MongoDB 데이터베이스 환경이 MariaDB 데이터베이스보다 빠르게 침입 데이터를 탐지할 수 있는 이유는 MariaDB 데이터베이스는 데이터베이스의 동시 접근에 따른 성능 감소와 데이터의 ACID를 보장하기 위해 많은 자원이 소요되는 등의 이유로 인해 탐지 시간이 증가하기 때문이다. MariaDB는 ACID를 보장함으로써 데이터베이스의 트랜잭션을 안정적으로 수행하지만 빅데이터 환경에서 많은 속도 저하가 발생한다. 또한, 분산 노드가 증가할수록 침입 데이터 탐지 시간이 증가하는데 보안 이벤트 저장과 마찬가지로 미들웨어 단에서 데이터베이스 서버에 있는 데이터를 검색할 때 데이터베이스의 ACID를 보장하기 위해 Spider Storage Engine과 데이터베이스 서버 간의 주고받는 데이터가 증가하기 때문이다. 그리고 분산 MongoDB 환경에서 가상 머신의 수가 증가함에 따라 속도가 개선되는 이유는 침입 데이터 탐지를 위한 패턴 매칭 작업을 분산 시켜 각 노드에서 처리할 수 있기 때문이다.

5. 결 론

본 논문에서는 매년 증가하고 있는 사이버 위협을 대비하기 위하여 다양한 장비에서 발생하는 빅데이터 보안 이벤트를 신속하게 저장하고 분석할 수 있는 분산 MongoDB 데이터베이스를 사용하는 침입탐지시스템을 제안하였다. 본 논문에서 제안한 시스템을 설계하고 구축하여 기존의 관계형 데이터베이스 기반의 시스템과 성능을 비교한 결과 보안 이벤트 저장의 경우 최대 60%, 그리고 침입 데이터 탐지는 최대 100% 성능 향상을 보였다. 이를 통해 분산 MongoDB 데이터베이스 기반의 침입탐지시스템 성능이 우수한 것을 확인할 수 있었다. 따라서, 빅데이터화 되는 보안 이벤트를 대비하여 침입탐지시스템을 도입할 경우 관계형 데이터베이스 기반 보다 분산 MongoDB 데이터베이스 기반의 침입탐지시스템을 구축할 시 보다 빠르게 침입 데이터에 대응할 수 있다.

또한 클라우드 환경에서 네트워크 규모나 데이터의 유입량의 변동에 따라 유동적으로 데이터베이스 서버의 수를 증감하며 사용하는 것이 의미가 있음을 확인하였다.

MongoDB 데이터베이스는 진입장벽이 낮고 비교적 간단한 형태의 함수 사용을 통해 기존의 SQL을 대체 할 수 있다. 이와 더불어 데이터 저장을 위해 JSON 구조를 사용하기 때문에 데이터를 직관적으로 파악할 수 있으므로 관계형 데이터베이스기반의 침입탐지시스템을 사용해오던 보안 관리자 또는 시스템 엔지니어가 큰 어려움 없이 기존 시스템을 변경해서 사용할 수 있을 것으로 기대된다.

향후 연구에서는 머신러닝을 이용한 탐지 룰 학습을 통해 오용 탐지 침입탐지시스템의 오탐률과 미탐률을 개선할 수 있는 방법에 대한 연구를 계획하고 있다.

References

[1] "IoT 2020 : Smart and Secure IoT Platform," International Electrotechnical Commission, pp.1-181, 2016.

[2] M. Chen, S. W. Mao, and Y. H. Liu, "Big Data : A Survey", Mobile Networks and Applications, Vol.19, Issue 2, pp. 171-209, Jan. 2014.

[3] Rehman, and Rafeeq Ur, "Intrusion Detection Systems With Snort: Advance IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID," Prentice Hall, 2003.

[4] ATEZENI, "Relational Database Theory," Addison Wesley Longman, 1993.

[5] H. J. Han, J. W. Kang, Y. H. Jung, and Y. W. Kim, "NoSQL-Based Distributed Processing System for Processing BigData Security Event," *2017 Spring Conference Proceedings*, Vol.24, Issue 1, Korea Information Processing Society.

[6] MariaDB [Internet], https://mariadb.org/

[7] MariaDB Spider Storage Engine [Internet], https://mariadb.com/kb/en/library/spider/

[8] M. Armbrust, et al., "Above the Clouds : A Berkeley View of Cloud Computing," *Electrical Engineering and Computer Sciences University of California at Berkeley*, 2009.

[9] MongoDB [Internet], https://www.mongodb.org

[10] Shannon Bradshaw, "Mongodb : The Definitive Guide : Powerful and Scalable Data Storage," O'ReillyMedia, 2017.

[11] M. H. Kang, "Completion of IDS and Security Control by Big Data Analysis," Wowbooks, 2013.

[12] J. Beale, A. R. Baker, B. Caswell, "Snort : IDS and IPS Toolkit," Syngress, 2007.

[13] G. Serpen and E. Aghaei, "Host-based Misuse Intrusion Detection using PCA Feature Extraction and kNN Classification Algorithms," *Intelligent Data Analysis*, Vol. 22, No.5, pp.1101-1114, 2018.

[14] S. Aljawarneh, M. Aldwairi, and M. B. Yassein, "Anomaly-based Intrusion Detection System Through Feature Selection Analysis and Building Hybrid Efficient Model," *Journal of Computational Science*, Vol.25, pp. 152-160, 2018.

[15] Y. W. Kim and S. Y. Lee, "Analysis and Understanding of Cloud Computing", Information and Communication on April, The Korean Institute of Communication and Information Sciences, pp. 87-92, 2015.

[16] G. Lu and W. H. Zeng, "Cloud Computing Survey," *Applied Mechanics and Materials*, Volume 530-531, pp. 650-661, 2014.

[17] OpenStack, [Internet], https://www.openstack.org/

[18] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection," *Compute & Security*, Vol.31, Issue 3, pp.357-374, May 2012.

한 효 준

https://orcid.org/0000-0003-4073-620X
e-mail : han6343@gmail.com
2016년 동국대학교 정보통신공학과(학사)
2018년 동국대학교 정보통신공학과(석사)
2018년 ~ 현 재 동국대학교
정보통신공학과 박사과정
관심분야 : 클라우드 컴퓨팅, 분산처리 시스템, 침입탐지시스템

김 혁 호

https://orcid.org/0000-0003-2730-471X
e-mail : hyukho.kim@gmail.com
2003년 대전대학교 정보통신공학과(학사)
2005년 동국대학교 정보통신공학과(석사)
2011년 동국대학교 정보통신공학과(박사)
2011년 ~ 현 재 삼성전자 종합기술원 전문연구원
관심분야 : 클라우드 및 그리드 컴퓨팅, 분산 컴퓨팅, HPC 시스템

김 양 우

https://orcid.org/0000-0001-5525-9472
e-mail : ywkim@dongguk.edu
1984년 연세대학교 전자공학과(공학사)
1986년 Syracuse Univ. 컴퓨터공학전공 (공학석사)
1992년 Syracuse Univ. 컴퓨터공학전공 (공학박사)
1992년 ~ 1996년 한국전자통신연구원 선임연구원
1996년 ~ 현 재 동국대학교 정보통신공학과 교수
관심분야 : 클라우드 및 그리드 컴퓨팅, 분산컴퓨팅 시스템