논문 2021-16-13

쿠버네티스 API server의 Transport Layer Security 패킷 실시간 복호화 및 시각화 시스템

(API Server Transport Layer Security Packets Real-Time Decryption and Visualization System in Kubernetes)

김 태 현, 김 태 영, 최 미 희, 진 성 근*
(Tae-Hyun Kim, Tae-Young Kim, Me-Hee Choi, Sunggeun Jin)

Abstract : The cloud computing evolution has brought us increasing necessity to manage virtual resources. For this reason, Kubernetes has developed to realize autonomous resource management in a large scale. It provides cloud computing infrastructure to handle cluster creations and deletions in a secure virtual computing environment. In the paper, we provide a monitoring scheme in which users can observe securely encrypted protocols while each Kubernetes component exchanges their packets. Eventually, users can utilize the proposed scheme for debugging as well as monitoring.

Keywords : Kubernetes, API access control, Cloud computing, Transport Layer Security, SSLKeyLog, Decryption

Ⅰ. 서 론

컨테이너 가상화 기술이 클라우드 컴퓨팅 환경에 활용되고 컨테이너를 통하여 서비스를 제공하는 노드의 수가 기하급수적으로 증가하면서 클라우드 컴퓨팅을 관리하고 운용하는 부담도 함께 늘어나게 되었다. 그러므로, 클라우드 컴퓨팅 환경에서 관리자의 인위적인 개입을 최소화하고 자동화된 방법으로 컨테이너를 관리하는 관리도구의 필요성이 제기되었다. 이러한 필요성을 만족하기 위하여 개발된 구글의 쿠버네티스는 사실상의 표준 기술로써 클라우드 컴퓨팅 환경에 활용되고 있다. 쿠버네티스는 여러 개의 컴퓨팅 노드를 하나의 클러스터로 묶어 관리하면서 자원을 효율적으로 활용하여 컨테이너를 배치하거나 관리할 수 있도록 다양한 알고리즘과 배치 방법을 제공한다. 이를 이용하여 사용자도 쉽고 빠르게 클라우드 컴퓨팅 환경에서 컨테이너를 관리할 수 있다. 쿠버네티스의 API server는 클러스터 내의 컴퓨팅 자원을 파악하고 각 노드에 컨테이너를 배치 및 관리하는 기능을 담당하여 쿠버네티스에서 중심적인 역할을 수행한다.

APIserver는 클러스터 내의 모든 제어 및 결정 권한을 보유하고 있으며 API server의 결정에 따라 클러스터의 작업이 할당된다. 그러므로 API server를 위한 보안은 매우 중요하다. 쿠버네티스는 API server의 보안을 위해 세가지 인증 단계를 구현하여 안전한 보안체계를 구성한다. 이를 위하여 클러스터 내의 대부분 통신은 TLS (Transport Layer Security)로 암호화한다. TLS는 컴퓨터 네트워크 통신에 보안을 제공하기 위해 설계된 암호화 프로토콜이다 [1-3]. 그러나, 쿠버네티스 응용 프로그램 개발이나 쿠버네티스 시스템 관리를 위하여 암호화된 TLS 패킷에 대한 복호화가 필요한 경우가 있다. 이렇게 복호화된 TLS 패킷은 개발자 혹은 관리자에게 제공되어 사용자 편의성을 향상하는데 활용될 수 있다.

우리는 API server의 보안을 해치지 않으면서 TLS 패킷을 복호화하고 이를 시각화된 정보로 사용자에게 제공하는 시스템을 개발하였다. 우리가 개발한 시스템은 다음과 같은 절차를 통하여 동작한다. (1) 우리의 API server의 보안을 해치지 않고 패킷을 복호화하는 기술을 기반으로 쿠버네티스 API server의 패킷을 복호화한다. (2) 그 후, 실시간으로 복호화된 패킷을 DB (Database)에 저장하도록 하고, (3) 저장된 데이터를 시각화하여 사용자에게 제공한다. 우리가 개발한 도구를 이용하여 클라우드 컴퓨팅 환경을 관리하는 관리자나 혹은 개발자가 향상된 GUI (Graphical User Interface)를 통하여 쉽게 문제점을 파악할 수 있다.

본 고에서는 다음과 같이 구성되어 있다. 2장에서는 쿠버네티스에 대해 간략히 설명하고 쿠버네티스 환경의 API server 접근을 위한 보안 인증 3단계에 대해 설명한다. 3장에서는 2장에서 설명한 인증 및 승인 과정이 TLS 프로토콜로 구성되어 클러스터 내부의 암호화된 TLS 통신에 대해 설명하고 암호화된 데이터 복호화 방법에 대해 제시한다. 그리고 해당 과정을 실시간으로 시각화하는 방법 및 시스템의 구조에 대해 설명한다. 4장에서는 암호화된 패킷을 실시간으로 복호화하는 환경을 실제로 구현하여 보여주고 5장에서 결론을 맺는다

*Corresponding Author (sgjin@daegu.ac.kr)
Received: Jan. 5, 2021, Revised: May 3, 2021, Accepted: May 11, 2021.
T.H. Kim: Daegu University (M.S.)
T.Y. Kim: Daegu University (M.S.)
M.H. Choi: Daegu University (B.S.)
S. Jin: Daegu University (Assoc. Prof.)
※ 이 성과는 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.2016R1D1A1B04932067).

 2021 Jun. 16(3) 99-105
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2021.16.3.99

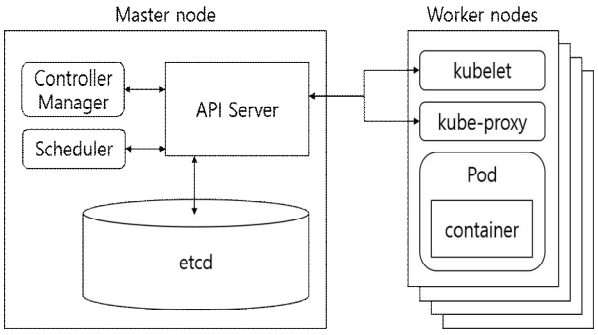

그림 1. 쿠버네티스 구조
Fig. 1. Architecture of kubernetes

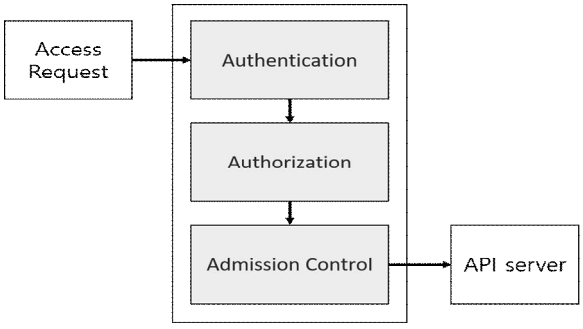

그림 2. API server 접근 인증 단계
Fig. 2. API server access step

Ⅱ. 배 경

1. 쿠버네티스의 구조

쿠버네티스의 구조는 그림 1과 같다 [4].

노드의 역할을 기준으로 마스터노드와 워커노드 분류된다. 마스터 노드는 쿠버네티스 클러스터 내의 워커 노드에 컨테이너 배치, 각 컨테이너 및 노드들 간의 통신 설정, 유지관리 등 클러스터 내부의 제어 역할을 담당한다. 워커노드는 마스터노드가 할당한 작업들을 kubelet을 통해 전달받으며 이 때 전달받은 작업을 실행한 후 마스터노드에 정상 작동 여부를 반환하고 마스터노드는 보고받은 정보를 바탕으로 최적의 상태를 위한 작업을 결정하여 재할당한다 [5]. 이와 같이 마스터노드와 워커노드가 하나의 클러스터로써 최적의 상태를 유지하도록 구조화되어 있으며 이 모든 결정을 판단하는 것이 마스터노드의 API server이다. API server는 클러스터 내의 모든 제어 및 결정 권한을 갖고 있다. 그러므로 관리자는 API server를 통해 새로운 작업을 할당할 수 있으며 클러스터를 관리하는 각종 정책에도 직접 관여할 수 있다. 이때 사용하는 것이 kubectl 이며, 클라이언트 라이브러리나 REST 요청을 통해서도 API에 접근할 수 있다. 쿠버네티스 클러스터는 443 포트를 통해 API server를 제공하고 있다. 해당 과정에서 API server는 3가지 인증 단계를 거치며 접근하도록 보안체계가 구성되어 있다 [6, 7]. 3가지 인증 단계는 그림 2와 같다.

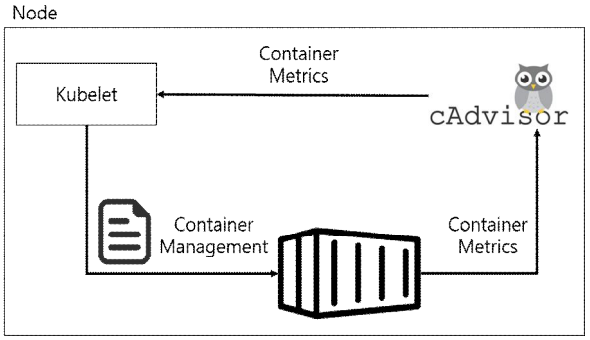

그림 3. Kubelet을 통한 작업 할당
Fig. 3. Allocating jobs through kubelet

첫 번째는 인증 (Authentication)이라는 단계이다. 인증이란 API server에 접근하려는 사람이 누구인지를 시스템이 확인하는 절차이다. 쿠버네티스에서는 인증을 위한 다양한 방법이 존재한다 [8]. 쿠버네티스 관리자들은 일반적으로 kubectl을 통해 API server에 접근한다. 이때 쿠버네티스는 암호화 통신을 위해 PKI (Public Key Infrastructure) 방법을 사용한다 [9, 10]. PKI란 공개키 기반 인증 방법으로 X.509라는 표준 포맷을 통해 암호화된 통신뿐만 아니라 인증에도 사용할 수 있다. 그 외에도 proxy 서버를 통해 대리 인증을 할 수도 있고, HTTP Authentication, OpenID Connect, 웹훅 등 다양한 방법으로 인증할 수 있다. 또한, 쉽게 인증 체계를 확장시킬 수 있으며 이는 사용자 인증 방식에 대해 자유롭고 유연하도록 개발했다고 볼 수 있다. 인증에 사용되는 모듈은 여러 개를 지정할 수 있고, 인증 모듈이 여러개인 경우 각 모듈을 순차적으로 진행한다. 요청된 인증이 실패하는 경우 HTTP 상태코드 401을 수신한다.

인증 단계를 통해 사용자를 신뢰할 수 있는지 확인이 되었다면 인가 단계로 넘어간다. 인가 (Authorization) 단계에서는 인증된 사용자가 소유한 권한확인 및 요청된 작업의 실행 가능 여부를 확인한다. 예를 들면, 사용자의 create 권한만을 소유하고 있는데 get 요청을 하는 경우 사용자의 인가는 거부된다. 인가 거부 HTTP 상태코드는 403이다. 인가 모듈도 여러 개를 구성할 수 있으며 쿠버네티스에서는 ABAC (Attribute-based access control), RBAC (Role-based access control), 웹훅 등의 인가 모듈을 지원한다. 인가 방식은 --authorization-mode 옵션을 사용하여 지정할 수 있으며, 모든 요청을 차단하거나 허용 또는 ABAC, RBAC, 웹훅 등 사용할 모듈을 직접 선택할 수 있다. 승인 제어 (Admission Control)란 모든 요청을 수정하거나 승인, 거부하는 역할을 한다. 즉, 요청을 실행하기 이전 마지막으로 요청 내용에 대해 승인 여부를 결정하는 것이다. 쿠버네티스 관리자는 승인 제어 모듈을 지정할 수 있으며, 이를 통해 모든 승인을 통과 또는 거부하거나 자원 할당량에 따라 결정, 서비스 계정을 통한 자동화, 승인 결정 여부를 웹훅 백엔드에 연결 등 다양한 방법으로 승인을 제어할 수 있다.

API server에 3가지 단계를 모두 거친 뒤 요청된 작업이

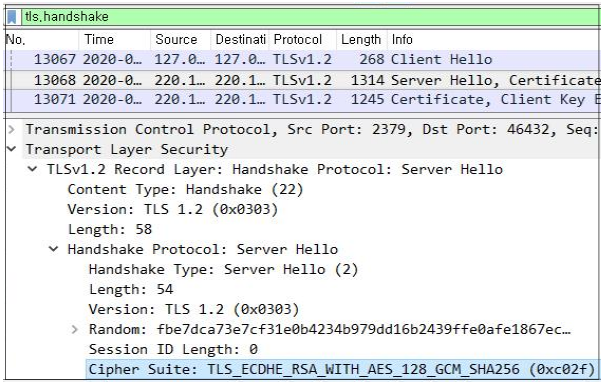

그림 4. 쿠버네티스 TLS 패킷의 암호화 스위트
Fig. 4. Cipher suite of kubernetes TLS packet

승인되면 API server는 각 노드의 kubelet을 통해 작업을 할당한다 [11]. Kubelet은 에이전트와 같은 역할을 수행하며 여러 노드가 하나의 클러스터로 통합되도록 해준다. Kubelet의 동작 과정은 그림 3과 같다. Kubelet은 각 노드 내에서 데몬으로 실행되며 각 노드들의 자원 (CPU, memory, volume)들을 클러스터의 일부가 될 수 있도록 한다. 클러스터 내 각 노드들은 kubelet을 통해 통신이 이뤄지고, 쿠버네티스는 보안을 위해 대부분의 통신이 TLS 암호화 네트워크로 통신되도록 하였다. 쿠버네티스는 3가지 인증 단계 및 대부분의 통신을 암호화함으로써 안전한 클러스터 보안체계를 제공하고 있다.

Ⅲ. 쿠버네티스 패킷 복호화

그림 4는 쿠버네티스의 클러스터에서 패킷을 캡쳐한 것이다. 해당 패킷에 사용된 암호화 스위트 (Cipher Suite)는 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256라는 암호화 스위트를 사용한다. 암호화 스위트는 다음과 같이 이루어져 있다.

(프로토콜)_(키 교환방식)_(인증서 검증)_WITH_(대칭 암호 알고리즘)_(블록 암호 운용방식)_(해시 알고리즘)

즉, 그림 4에서 사용된 암호화 스위트는 TLS 프로토콜을 사용하며 ECDHE 키교환 방식, RSA 알고리즘을 통한 인증, 암호화 블록의 크기가 128비트인 AES 대칭 암호알고리즘을 사용하고, 블록 암호 운용모드는 GCM, HMAC용 해시 알고리즘은 SHA-256을 사용한다는 의미이다. ECDHE 알고리즘을 사용한다는 점이 주목해야 할 부분이다. ECDHE (Elliptic Curve Diffie-Hellman Exchange)란 타원 곡선 디피-헬만 키 교환으로 송수신자가 서로 같은 키를 별도의 비밀키 교환 없이 공유하기 위해 만들어진 알고리즘이며, 이를 타원곡선의 수학적 성질을 이용한 암호화 알고리즘이다 [12]. 중요한 것은 ECDHE의 마지막 E가 Exchange가 아닌 Ephemeral이라는 의미로도 쓰이며 이는 임시적이라는 의미로 TLS 통신 시 암호화에 세션키라는 것을 사용하며 이는 매우 짧은 주기로 변경되기 때문에 임시적인 성질을 가지고

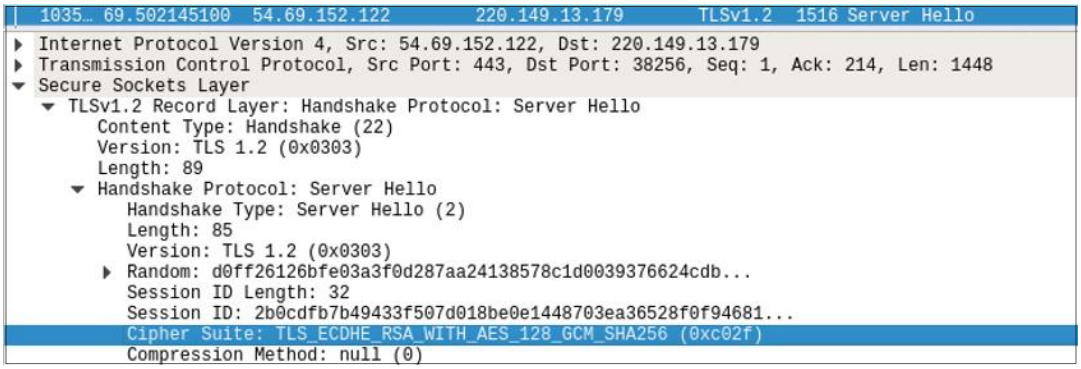

그림 5. Firefox TLS 패킷의 암호화 스위트
Fig. 5. Cipher suite of firefox TLS packet

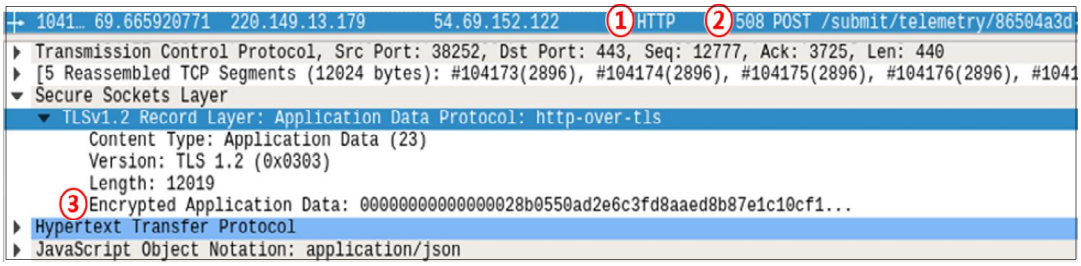

그림 6. 복호화된 firefox TLS 패킷
Fig. 6. Decrypted firefox TLS packets

있다. 그러므로 공격자가 특정 패킷의 비밀키를 찾아내더라도 임시적인 특성을 갖는 세션키를 사용하기 때문에 해당된 짧은 순간의 패킷만 복호화 가능하며 다른 패킷들은 지속적으로 보호된다. 또한, 세션키를 사용하여 암호화 및 복호화를 하기 때문에 서버와 클라이언트의 비밀키 만으로는 복호화 할 수 없다. 세션키는 매 연결 마다 랜덤으로 생성된다.

그림 5는 Firefox의 'Server Hello' 패킷을 캡쳐한 것이고, 그림 6은 TLS handshake 과정을 모두 마친 뒤 교환한 암호화된 어플리케이션 데이터를 복호화 한 그림이다. 그림 6의 ①을 보면 해당 패킷이 HTTP 프로토콜을 사용하는 것을 확인할 수 있고, ②를 통해 복호화된 데이터를 확인할 수 있다. 또한, 해당 패킷은 암호화된 패킷을 복호화한것이므로 그림 6의 ③에서 원래의 암호화된 Encrypted Application Data도 동시에 확인할 수 있다.

Firefox에서 암호화된 패킷을 복호화하는 방법은 SSLKeyLog를 사용하여 복호화하는 방법을 사용한다. SSLKeyLog란 주기적으로 변경되는 세션키를 모두 기록한 파일이다. 세션키는 TLS handshake시 생성되는데, 매 연결시 마다 세션키가 임시로 생성되기 때문에 해당 세션키를 모두 기록하면 암호화된 패킷을 복호화 하는데 사용할 수 있다.

그림 4의 쿠버네티스 암호화 스위트는 그림 5의 Firefox 패킷의 암호화 스위트와 동일한 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256을 사용한다. SSLKeyLog 표준은 NSSKeyLog형식 [13]을 따른다. 그림 7에 따른 NSSKeyLog 형식은 다음과 같다.

CLIENT_RANDOM <32byte의 ClientRandom> <48byte의 Pre-master secret>

ClientRandom은 32byte의 난수이며 암호화를 위한 키를 생성하는데 사용된다. Pre-master secret은 48byte이며 비밀키인 master secret을 복호화하는데 사용되는 secret이다. Master secret는 pre-master secret과 임의 값(Client/Server Random)을 사용하여 PRF (Pseudo Random Function)으로 복호화하여 사용할 수 있다 [14]. 쿠버네티스에서도 그림 4, 5, 6과 같이 Firefox와 같은 암호화 스위트를 사용하므로, 모든 TLS 패킷을 복호화하기 위해 클러스터의 제어판과 같은 API server에 세션키를 기록하는 TLSKeyLog 기능을 추가하였다.

그림 8은 쿠버네티스 패킷을 TLSKeyLog를 통해 복호화한 모습이다. ①에 표시된 Data의 정보는 아래쪽에서 자세히 확인 가능하다. 파란색으로 칠해진 부분이 Data에 해당하는 부분이며 ②는 Data의 헥스값, ③은 문자열의 형태로 나타낸 것이다.

해당 기능을 실시간으로 실행하기 위해 그림 9와 같은 구조를 설계하였다. 우선 API 서버의 패킷을 주기적으로 수집하고 매 연결시 마다 세션키를 파일로 기록하도록 하였다. PacketDecryptor는 이전 과정에서 기록된 세션키를 사용하여 암호화된 패킷을 복호화한다. 복호화된 패킷은 데이터베이스에 실시간으로 저장되며, 복호화된 데이터는 사용자 편의에 맞게 가공하여 시각화할 수 있다.

```
CLIENT_RANDOM 5f4dad779789bc5142cacf54f5dafba0a06235640796f400
48ce4d0d1df63ad8 a4d69a3fa4222d6b6f2492e66dca2b1fc4e2bc143df849
ad45eff9f43650cc2a2e28a58873b6812f2e3b238a695085ad
```

그림 7. SSLKeyLog 예시
Fig. 7. SSLKeyLog example

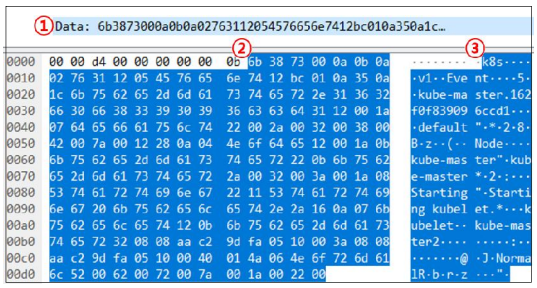

그림 8. 복호화된 쿠버네티스 TLS 패킷
Fig. 8. Decrypted kubernetes TLS packet

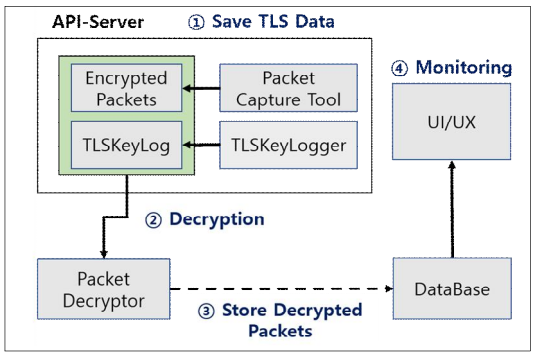

그림 9. 실시간 복호화 및 시각화 시스템 구조
Fig. 9. Architecture of real-time decryption and visualization system

Ⅳ. 실험 및 결과

실험환경은 그림 9와 같다. API server의 패킷을 수집하고, 수집된 패킷들은 TLSKeyLog를 통하여 일정 주기로 복호화되어 DB에 저장된다. 저장된 DB의 데이터는 시각화 도구에서 가공되어 사용자에게 보여지고, 이를 통해 사용자는 복호화된 데이터를 UI (User Interface)로 쉽게 확인할 수 있다.

그림 10의 명령어 'kubectl get pods'는 kubectl을 사용하여 default 네임스페이스의 Pod 리스트를 가져오는 것이다. 해당 명령어를 실행하면 클라우드 관리자는 kubectl 명령어를 통해 API server에 접근하고 이 과정은 TLS 프로토콜을 통해 이루어진다. 이 때 TLS를 복호화하기 위한 세션키가 TLSKeyLogger를 통해 기록된다. 기록된 세션키는 Packet Decryptor를 통해 실시간으로 데이터베이스에 저장된다.

그림 11은 DB에 저장된 복호화된 패킷들이다. 데이터베이스에 저장된 데이터를 살펴보면 "Time"이 그림 10과 동일한 것을 알 수 있다. 즉, 복호화된 데이터들은 모두 실시간으로 저장되며 소수점 6자리까지의 초 단위로 기록된다.

```
root@kube-master:~# date; kubectl get pods
Wed 30 Dec 2020 06:57:25 PM KST
NAME                     READY   STATUS    RESTARTS   AGE
grafana-tklfl            1/1     Running   0          39m
mongo-controller-xhs26   1/1     Running   0          38m
web-controller-4gm6t     1/1     Running   0          45m
root@kube-master:~#
```

그림 10. Pod 리스트를 호출하는 kubectl 명령어
Fig. 10. Kubectl command to get Pod list

{ "_id" : ObjectId("5fec4f0e92107d00301fd9eb"), No : 7272, "Time" : "2020-12-30 09:57:25.910679", "Source" : "220.149.11.96", "Destination" : "220.149.11.96", "Protocol" : "HTTP2", "Length" : 99, "Info" "DATA(71) (application/json)", "Data" : "{\"kind\":\"Table\",\"apiVersion\":\"meta.k8s.io/v1\",\"metadata\":{\"selfLink\":\"/api/v1/namespaces/default/pods\",\"resourceVersion\":\"15522\"}Xcolumn Defi nitions\":[{\"name\":\"Name\",\"type\":\"string\",\"format\":\"name\",\"description\":\"Name must be u nique within a namespace. Is required when creating resources, although some resources may allow a cli ent to request the generation of an appropriate name automatically. Name is primarily intended for cre

그림 11. DB에 저장된 복호 패킷
Fig. 11. Decrypted packets stored in DB

2020-12-30 09:57:30…	220.149.11.96	220.149.11.96	HTTP2	140	HEADERS[1]: GET /healthz	
2020-12-30 09:57:30…	220.149.11.96	220.149.11.96	HTTP2	154	Magic, SETTINGS[0], WINDO…	
2020-12-30 09:57:25…	220.149.11.96	220.149.11.96	HTTP2	99	DATA[71] (application/jso…	{"kind":"Table","apiVe
2020-12-30 09:57:25…	220.149.11.96	220.149.11.96	HTTP2	103	WINDOW_UPDATE[71]	
2020-12-30 09:57:25…	220.149.11.96	220.149.11.96	HTTP2	10614	DATA[71]	{"kind":"Table","apiVe
2020-12-30 09:57:25…	220.149.11.96	220.149.11.96	HTTP2	103	HEADERS[71]: 200 OK	

그림 12. 실시간 복호화 및 시각화 도구 UI
Fig. 12. Real-time decryption and visualization tool UI

해당 데이터는 시각화 도구를 통해 JSON 형식의 데이터를 UI에 맞게 가공하였으며, UI는 그림 12와 같이 패킷이 수집된 시간 (Time), 출발지와 도착지의 IP 주소 (Source/Destination), 프로토콜 (Protocol), 패킷이 가진 정보 (Info), 복호화된 데이터 정보 (Data)로 구성된다.

그림 12의 데이터를 자세히 보려면 해당 데이터를 클릭하여 그림 13과 같이 더욱 자세하게 확인할 수 있다. 또한 그림 13의 metadata의 selfLink란 API server에 접근할 수 있는 REST-API 형식이며 default 네임스페이스의 Pod 리스트를 가져오는 'kubectl get pods'와 같다고 할 수 있다.

selfLink: "/api/v1/namespaces/default/pods"

그림 14는 kubectl의 proxy기능을 통해 REST-API로 API server에 접근한 그림이며 가공되지 않은 'kubectl get pods' 형태의 모습이다. 해당 방법을 통해 API server에 접근하게 되면 결과값은 그림 10과 동일하게 네임스페이스가 default인 Pod의 리스트를 가져오는 것을 확인할 수 있다.

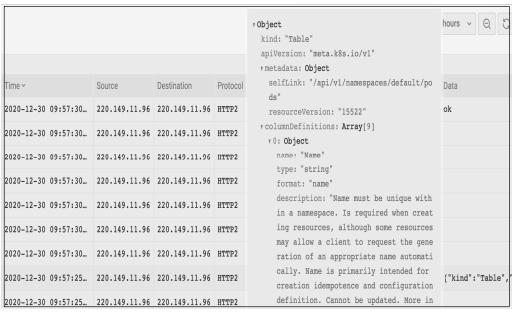

그림 13. 시각화 도구로 수집한 Pod 리스트 데이터
Fig. 13. Pod list data collected by the visualization tool

root@kube-master:~# curl 220.149.11.96:9999/api/v1/namespaces/default/pods
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/namespaces/default/pods",
    "resourceVersion": "180752"
  },
  "items": [
    {
      "metadata": {
        "name": "grafana-tklfl",
        "generateName": "grafana-",
        "namespace": "default",
        "selfLink": "/api/v1/namespaces/default/pods/grafana-tklfl",
        "uid": "5a915888-59ef-4aa0-9075-9fca8cb04476",
        "resourceVersion": "10176",
        "creationTimestamp": "2020-12-30T09:17:54Z",
        "labels": {
          "name": "grafana"
        },
        "ownerReferences": [
          {
            "apiVersion": "v1",
            "kind": "ReplicationController",
            "name": "grafana",
            "uid": "032e249d-d28a-43ac-8deb-30f6389afcdd",
            "controller": true,
            "blockOwnerDeletion": true
          }

그림 14. Pod 리스트를 호출하는 REST-API
Fig. 14. REST-API to get Pod list

V. 결 론

현재에도 암호 및 보안을 위한 기술은 계속해서 발전하고 있으며, 많은 컴퓨터 전문가들이 취약점과 보완할 점을 분석하고 있다. TLS에서 ECDHE 알고리즘은 보안을 위해 임시적인 특성을 갖는 비밀키인 세션키를 사용하기 때문에 중간에 다른 사용자가 패킷을 가로채더라도 디피-헬만 키 교환 알고리즘으로 인해 복호화를 위한 비밀키를 찾기도 힘들 뿐만 아니라 만약 해당하는 비밀키를 찾아내더라도 해당 패킷만 복호화 가능하고 다른 패킷은 각각 다른 세션키를 사용하므로 다른 암호화 알고리즘보다 안전하다. 이러한 특성 때문에 쿠버네티스에서도 해당 암호화 방식을 채택하여 패킷을 보호하고 있다. 그러나, 디버깅 또는 특별한 목적을 위해 ECDHE로 보안된 TLS를 분석해야 하는 경우 TLSKeyLog 기술을 통해 쉽게 복호화 및 분석할 수 있다. 복호화된 데이터는 주기적으로 DB에 저장하고 사용자는 시각화 도구를 통하여 실시간으로 복호화된 데이터를 확인할 수 있다. 또한, 데이터를 사용자에게 알맞은 형태로 가공하

여 UI를 구성할 수 있다.

본 고에서는 임시적인 세션키를 기록하여 수집된 TLS 패킷을 복호화하고 DB를 통해 저장 및 시각화하는 전체적인 구조와 과정에 대해 설명하였다. API server의 데이터를 보기 쉽게 실시간으로 시각화함으로써 클러스터 관리/운용자는 해당 기술을 활용하여 복호화된 데이터를 기반으로 쉽게 문제점을 파악할 수 있도록 추가 개발이 가능하다. 예를 들면, TLS 패킷 복호화 기술을 통해 특정 명령에 대한 패킷의 흐름을 패턴화하고, 또는 해당 데이터를 통해 다양한 오류 과정을 패턴화하여 문제 해결을 자동화할 수 있도록 추가 개발할 수 있다.

References

[1] Huang, L. S., Adhikarla, S., Boneh, D., Jackson, C, “An Experimental Study of TLS Forward Secrecy Deployments,” IEEE Internet Computing, pp. 43-51, 2014.

[2] Morrissey, P., Smart, N. P., Warinschi, B., “A Modular Security Analysis of the TLS Handshake Protocol,” International Conference on the Theory and Application of Cryptology and Information Security, pp. 55-73, 2008.

[3] Dierks, T., Rescorla, E., “The Transport Layer Security (TLS) protocol version 1.2,” 2008.

[4] https://kubernetes.io/

[5] Habbal, N., “Enhancing Availability of Microservice Architecture: A Case Study on Kubernetes Security Configurations,” 2020.

[6] He, X., Yang, X., “Authentication and Authorization of end user in Microservice Architecture,” Physics: Conference Series, Vol. 910, 2017.

[7] Shamim, M. S. I., Bhuiyan, F. A., Rahman, A., “XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices,” 2020 IEEE Secure Development (SecDev), pp. 58-64, 2020.

[8] R. Eidenbenz, Y. Pignolet, A. Ryser, "Latency-Aware Industrial Fog Application Orchestration with Kubernetes," 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC), pp. 164-171, 2020.

[9] Nash, A., Duane, W., Joseph, C., “PKI: Implementing and Managing E-security,” 2001.

[10] P. Szalachowski, L. Chuat, A. Perrig, “PKI Safety Net (PKISN): Addressing the Too-Big-to-Be-Revoked Problem of the TLS Ecosystem,” 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 407-422, 2016.

[11] T.H. Kim, T.Y. Kim, Y.E. Choi, M.H. Choi, Sunngeun Jin, “Virtualization and Kubernetes,”. OSIA Standards & Technology Review, pp. 4-10, 2020 (in Korean).

[12] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., VanderSloot, B., “Imperfect Forward Secrecy: How Diffie-Hellman fails in Practice,” 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 5-17, 2015.

[13] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

[14] Luby, M., Rackoff, C., “How to Construct Pseudorandom Permutations from Pseudorandom Functions,” SIAM Journal on Computing, pp. 373-386, 1988.

Tae-Hyun Kim (김 태 현)

2021 Computer Engineering from Daegu University(B.S.)

2021~ Computer Engineering from Daegu University, Master Student

Career:

2020~ Researcher of CraftX Inc.

Field of Interests: Cloud Computing, Cyber Security

Email: tolriaz@daegu.ac.kr

Tae-Young Kim (김 태 영)

2021 Computer Engineering from Daegu University (B.S.)

2021~ Computer Engineering from Daegu University, Master Student

Career:

2020~ Researcher of CraftX Inc.

Field of Interests: Cloud Computing, Internet of Things

Email: tkken12@daegu.ac.kr

Me-Hee Choi (최 미 희)

2018~ Computer Engineering from Daegu University, Bachelor Student

Field of Interests: Cloud Computing

Email: risio379@daegu.ac.kr

Sunggeun Jin (진 성 근)

1996 Depart of Electronic Engineering from Kyungpook National University (B.S.)

1998 Depart of Electronic Engineering from Kyungpook National University (M.S.)

2008 School of Electrical and Computer Engineering from Seoul National University (Ph.D.)

Career:

1998~2013 Electronics and Telecommunications Research Institute

2012~2013 Adjunct Professor, University of Science and Technology

2013~ Associate Professor, Daegu University

2020~ CEO of CraftX Inc.

Field of Interests: Cloud Computing, Edge Computing,

Email: sgjin@daegu.ac.kr