TC-BPF 기반 실시간 TLS 암호화 모니터링 시스템

차호현[1], 신주용[2], 남재현[3]
[1]단국대학교 모바일시스템공학과 학부생
[2]단국대학교 인공지능융합학과 석사과정
[3]단국대학교 컴퓨터공학과 교수

[1]outcider112@dankook.ac.kr, [2]juyongshin@dankook.ac.kr, [3]namjh@dankook.ac.kr

Runtime Cryptographic Monitoring System based TC-BPF

Hohyeon Cha[1], Juyong Shin[2], Jaehyun Nam[3]
[1]Dept. of Mobile Systems Engineering, Dankook University (Undergraduate student)
[2]Dept. of Artificial Intelligence Convergence, Dankook University (Graduate student)
[3]Dept. of Computer Engineering, Dankook University (Professor)

요 약

클라우드 도입이 확산됨에 따라 보안 위협 역시 지속적으로 증가하고 있다. 특히, 클라우드 시스템에서 데이터를 안전하게 관리하기 위해 TLS 프로토콜을 사용해 네트워크 연결을 암호화하지만, 관리자의 잘못된 설정으로 인해 데이터 유출의 위험이 존재할 수 있다. 본 연구에서는 TLS 프로토콜을 통해 암호화된 통신에서 사용되는 설정 정보를 확인하고, Traffic Control BPF (TC-BPF) 기술을 적용하여 현 클라우드 시스템 내 TLS 구성 정보를 수집하는 방법을 제안한다. 또한, 이를 바탕으로 수집된 정보를 기반으로 취약한 설정을 분석하고 로그를 생성하는 시스템을 구현하였다. 그 결과, 네트워크 지연 시간에 약 1.3%의 성능 저하만을 발생시켰으며, 수집된 로그를 기반으로 암호화 알고리즘의 취약성을 효과적으로 분류할 수 있었다.

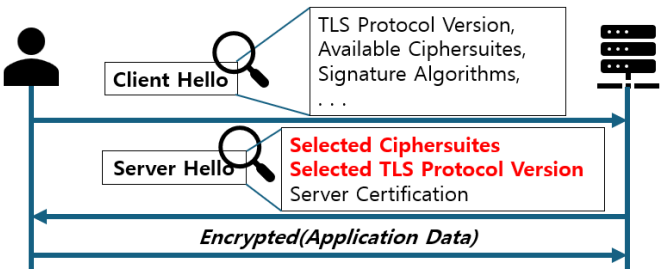
1. 서론

마이크로서비스 아키텍처(MSA)의 도입으로 인해 컨테이너와 API 간의 네트워크 기반 통신이 증가하였다. 이에 따라 서비스 간 데이터가 네트워크를 통해 노출되는 빈도도 높아졌으며, 이를 방지하기 위해 TLS 프로토콜을 사용하여 데이터를 암호화함으로써 제 3 자의 접근을 차단할 수 있다. 그러나 TLS 프로토콜 사용 시, 취약한 암호화 알고리즘을 선택하는 등의 문제로 인해 보안 취약점이 발생할 수 있다.

기존 솔루션 중 하나인 k8tls [1]는 각 서비스에 대해 TLS 구성 정보를 확인할 수 있으나, 서비스에 직접 클라이언트로서 TLS 연결을 수행하는 방식이다. 이는 마이크로서비스 간의 TLS 통신을 지속적으로 모니터링하기 어렵고, 클라이언트 구성 정보가 테스트 중인 TLS 구성 정보에 영향을 미쳐 실제 운영 환경에서 사용되는 TLS 구성 정보와 다를 수 있다.

따라서 본 연구에서는 컨테이너 기반 MSA 환경에서 각 서비스가 사용하는 TLS 구성 정보를 실시간으로 수집하고, 이를 통해 구성의 취약성을 분석하는 시스템을 제안한다. 제안된 시스템을 적용하여 수집된 로그를 바탕으로 성능 평가를 수행할 예정이다.

2. TLS 프로토콜



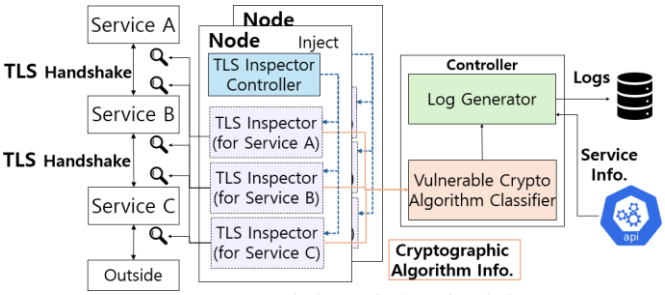
(그림 1) TLS 핸드쉐이크 다이어그램 및 교환되는 정보

TLS 프로토콜은 TCP 기반의 데이터 암호화 프로토콜로, 4 계층 위에서 동작한다. TLS 프로토콜 수행 시, 클라이언트는 사용 가능한 암호화 알고리즘이 포함된 '암호화 스위트' 후보군, '서명 알고리즘', 그리고 'TLS 프로토콜 버전'을 서버에 제시한다. 서버는 이에 대해 사용할 '암호화 스위트', CA 로부터 발급받은 '인증서', 그리고 선택된 프로토콜 버전 등의 정보를 평문으로 클라이언트와 교환한다. 이후, 양측은 선택된 '대칭키 알고리즘'과 '공개 키 알고리즘'을 사용하여 이후 어플리케이션 데이터 전송 시 해당 데이터를 암호화한다.

3. Linux Traffic Control 과 eBPF

Traffic Control 은 리눅스 커널에서 네트워크 트래픽을 관리하고 제어하는 도구로, 네트워크 네임스페이스 간의 트래픽에 대해 모니터링, 분류, 그리고 스케줄링 기능을 제공한다[2]. eBPF 는 이러한 기능을 확장하여, 커널의 수정을 가하지 않고도 사용자 정의 함수를 커널 내에서 샌드박스 환경에서 실행할 수 있게 한다[3]. 본 연구에서는 eBPF 기술을 통해 TLS 핸드쉐이크 패킷을 실시간 모니터링을 위해 활용한다.

4. 실시간 TLS 암호화 모니터링 시스템



(그림 2) 모니터링 아키텍처 제안

본 시스템은 클러스터 내에서 중앙의 Controller 가 각 노드에 TLS Inspector Controller 를 배포하고, 이 TLS Inspector Controller 는 각 서비스에 TLS Inspector 를 배포하는 구조로 동작한다. TLS Inspector 는 각 서비스의 TLS 통신에 대한 정보를 수집하며, 중앙의 Controller 는 이를 통합하여 처리한다. 각 서비스에 배포된 TLS Inspector 는 해당 서비스의 Network Namespace 에 eBPF 프로그램을 부착하여 TLS 핸드쉐이크 패킷으로 부터 IP 주소, 포트 번호, 암호화 스위트 등의 정보를 실시간으로 수집한다.

Controller 는 TLS Inspector 가 수집한 암호화 스위트, TLS 버전, 서명 알고리즘 등의 정보를 분석하고, 이를 바탕으로 암호화 수준을 등급별로 구분한다. 예를 들어, 취약한 암호화 알고리즘[4]을 사용하거나 TLS 1.2 미만의 프로토콜 버전을 사용하는 경우, 'Weakly Encrypted'로 TLS 프로토콜을 사용하지 않는 경우 'Not Encrypted'로 기록한다. 이후 IP 주소와 포트 정보를 기반으로 실질적인 서비스에 대한 정보를 도출하고, TLS 구성 정보와 병합하여 최종 로그로 생성한다.

5. 시스템 성능 평가 및 검증

제안된 시스템에 대한 성능 평가를 위해 <표 1>과 같은 환경에서 wrk 도구를 통해 Nginx 성능 변화를 측정하였다. 그리고 <표 2>는 TLS Inspector 의 적용 전/후에 대한 통신 성능을 나타내며, (그림 3)은 제안된 시스템에서 생성한 로그의 예시를 보여준다.

<표 1> 실험 환경

종류	Client (wrk)	Server (Nginx)
vCPU	4 Core	8 Core
OS/SW	Ubuntu22.04	Ubuntu22.04, nginx/1.18.0

<표 2> TLS Inspector 사용에 따른 통신 성능 변화

적용 전		적용 후	
지연시간(us)	처리량(MiB/s)	지연시간(us)	처리량(MiB/s)
671.15	54.96	680.09	54.47

```
Encryption Status : Weakly Encrypted
TLS VERSION : 1.2
Cipher Suite : TLS_RSA_WITH_AES_256_CBC_SHA
Certificate Signature : sha256WithRSAEncryption
```

(그림 3) 구현된 시스템에서 생성한 TLS 구성 로그

실험 결과, TLS Inspector 를 적용한 후 약 1.33%의 성능 저하는 발생하나, 구현된 시스템을 통해 실시간으로 TLS 구성 정보를 확인하고, 취약한 암호화 알고리즘의 사용 역시 감지할 수 있음을 확인하였다.

6. 결론

본 연구에서는 컨테이너로 격리된 분산 아키텍처(MSA) 환경에서 각 서비스가 사용하는 TLS 암호화 프로토콜의 구성 정보를 실시간으로 모니터링하는 시스템을 구현하였다. eBPF 를 활용하여 낮은 성능 저하와 함께 실시간 TLS 구성 정보 수집이 가능함을 확인하였다. 또한, 취약한 TLS 암호화 구성에 대한 탐지 역시 가능함을 확인하였다. 향후 연구로 취약한 TLS 암호화 구성 및 암호화 라이브러리의 동적 교체 시스템을 구현하여 본 시스템과 통합할 계획이다.

Acknowledgement

이 성과는 2024 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임. (RS-2023-00212738)

참고문헌

[1] Kubearmor, ‘k8tls’, accessed Sep. 17th, 2024, https://github.com/kubearmor/k8tls/

[2] Yang Zhou, Xingyu Xiang, Matthew Kiley et. al, ‘DINT: Fast In-Kernel Distributed Transactions with eBPF,’ USENIX, NSDI ’24.

[3] eBPF, ‘What is eBPF,’ accessed Sep. 17th, 2024, https://ebpf.io/what-is-ebpf/

[4] Elaine Barker, ‘NIST SP 800-131A: Transitioning the Use of Cryptographic Algorithms and Key Lengths’ Rev. 2, NIST, March 2019.