Software Design Specification

1. Introduction (소개)

1.1. Purpose (목적)

본 Software Design Specification(SDS) 문서는 데이터베이스 침입 탐지 시스템 (Database Intrusion Detection System, DB-IDS)의 상세 설계를 기술한다.

이 문서는 상위 문서인 Software Requirements Specification(SRS)을 기반으로 하며, 설계 및 구현 단계에서 참조할 수 있는 기준을 제공한다.

1.2. Scope (범위)

DB-IDS는 관계형 데이터베이스를 대상으로 프록시 계층에서 모든 SQL 쿼리를 모니터링하고, 패턴 기반/행동 기반/권한 기반 탐지를 통해 이상 행위를 식별한다.

탐지 이벤트 발생 시 관리자는 실시간 알림(Slack, 이메일)을 받고, 웹 기반 대시보드를 통해 로그와 이벤트를 시각화할 수 있다.

본 문서는 다음 설계 범위를 포함한다:

- 전체 시스템 아키텍처 및 컴포넌트 구조
- 데이터 저장소 설계
- 모듈별 상세 설계
- 사용자 인터페이스 설계
- 요구사항 추적 매트릭스

1.3. Definitions, acronyms and abbreviations (용어 및 약어 정의)

- IDS (Intrusion Detection System): 네트워크나 시스템에서 발생하는 이벤트를 모니터링하여 비인가된 접근이나 이상 행위를 탐지하는 시스템.
- DB-IDS (Database Intrusion Detection System): 데이터베이스 쿼리와 접근 패턴을 감시하여 침입 시도를 탐지하는 특화된 IDS.

- SQL (Structured Query Language): 관계형 데이터베이스 관리 시스템(RDBMS)에서 데이터 질의와 조작에 사용되는 표준 언어.
- SQL Injection: 사용자가 입력한 데이터를 검증하지 않고 SQL 문에 직접 포함시켜 실행하는 보안 취약점을 악용한 공격 기법.
- Proxy (프록시): 클라이언트와 서버 사이에서 통신을 중계하는 중간 계층. DB-IDS는 주로 프록시 기반으로 동작하여 쿼리를 가로채고 분석함.
- Pattern-based Detection (패턴 기반 탐지): 미리 정의된 공격 시그니처나 금지된 SQL 구문을 기반으로 탐지하는 기법.
- Behavior-based Detection (행동 기반 탐지): 정상 동작 프로파일과 비교하여 비정상적 행동(쿼리 빈도 급증, 데이터 반환량 폭증 등)을 탐지하는 기법.
- Authorization-based Detection (권한 기반 탐지): 특정 사용자나 계정이 데이터베이스 자원에 접근할 수 있도록 허용하는 과정.
- Alert (알림): 보안 이벤트가 발생했음을 관리자에게 통지하는 메시지. 이메일, Slack으로 전달 가능.
- Dashboard (대시보드): 관리자에게 탐지 이벤트, 로그, 통계 정보를 시각화하여 제공하는 UI.
- RDBMS (Relational Database Management System): 관계형 데이터 모델을 기반으로 데이터를 저장·관리하는 시스템. MySQL, PostgreSQL, Oracle 등이 포함됨.
- Admin (관리자): DB 침입을 탐지하기 위해 시스템을 사용하는 실 사용자.
- False Positive (오탐): 정상적인 쿼리를 침입으로 잘못 탐지하는 경우.
- False Negative (미탐): 실제 침입 시도를 탐지하지 못하는 경우.

1.4. References (참고자료)

- IEEE Std 830-1998, *Software Requirements Specification*
- IEEE Std 1016-1998, *Software Design Specification*
- Halmstad University (2022), *Anomaly Detection in SQL Databases*
- GreenSQL (오픈소스 DB 보안 프록시)
- OWASP SQL Injection Cheat Sheet
- ISO/IEC 27001:2013, *Information Security Management*

1.5. Overview (개요)

본 문서는 시스템의 아키텍처를 설명한 후, 각 컴포넌트의 설계를 상세히 기술한다. 이어서 데이터 설계, 사용자 인터페이스 설계, 요구사항 추적성, 그리고 부록을 포함한다.

이로써 개발자는 설계에 따라 코드를 작성할 수 있고, 테스터는 요구사항 충족 여부를 검증할 수 있으며, 운영자는 시스템의 구조적 특성을 이해할 수 있다.

2. System Overview (시스템 개요)

DB 침입 탐지 시스템(Database Intrusion Detection System, DB-IDS)은 데이터베이스 앞단 프록시로 동작하여 모든 SQL 쿼리를 감시하고, 다양한 탐지 기법을 통해 비정상적인 접근 시도를 식별한다.

시스템은 크게 세 가지 축으로 동작한다:

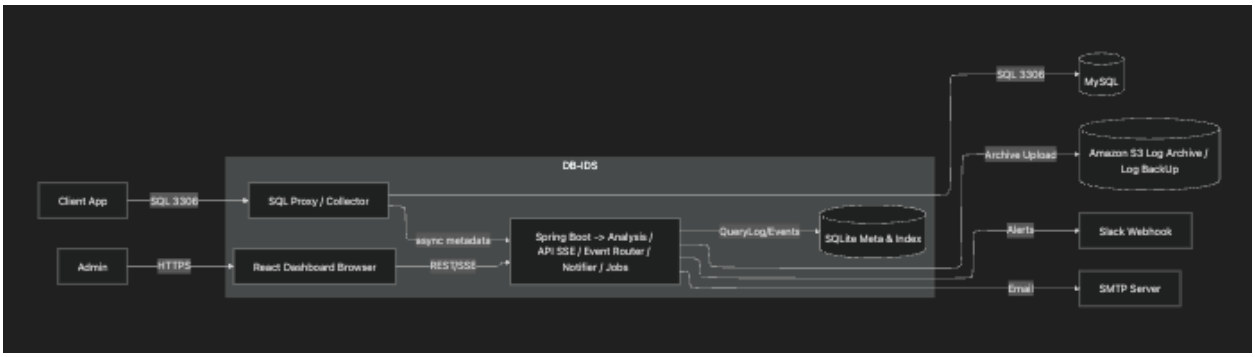
1. 수집(Collector): DB 서버로 전달되는 모든 SQL 요청과 응답을 가로채어 로그화한다.
2. 탐지 및 알람(Detection Engine & Alert): 패턴 기반, 행동 기반, 권한 기반 탐지 기법을 적용하여 비정상 행위를 판별하고 탐지 이벤트 발생 시 관리자에게 Slack/Email 알림을 전송한다.
3. 시각화(Dashboard): 웹 기반 대시보드를 통해 로그 및 이벤트를 시각적으로 제공한다.

DB-IDS는 MySQL 8.0을 기본 지원하며, Spring Boot 기반 탐지 엔진, React 기반 대시보드, SQLite 내부 저장소로 구성된다. 운영 환경은 Linux(Ubuntu 20.04+)이며, 클라우드 VM 또는 컨테이너 환경에서 배포 가능하다.

본 시스템을 통해 관리자는 단순한 DB 로그로는 식별하기 어려운 SQL Injection, 대량 조회, 비인가 권한 접근 등 이상 행위를 실시간으로 인지할 수 있으며, 데이터베이스 보안성을 크게 강화할 수 있다.

3. System Architecture (시스템 아키텍처)

3.1. Architectural Design (아키텍처 설계)



3.1.1. Context View

이 뷰는 DB-IDS 시스템을 블랙박스로 간주하고, 외부 액터 및 외부 시스템과의 관계를 보여준다. 시스템이 어떤 환경 속에 위치하며, 누가 이 시스템을 사용하고 어떤 외부 서비스와 통신하는지를 설명한다.

External Actors

- Client App (보호 대상 애플리케이션): SQL 쿼리를 DB에 전달하며, DB-IDS는 이 트래픽을 모니터링한다.
- Admin (운영자): React 기반 웹 브라우저를 통해 DB-IDS의 로그와 이벤트를 확인하고 설정을 제어한다.

External Systems

- MySQL 8.0 (모니터링 대상 DBMS): Client App의 SQL 요청을 실제로 처리하는 운영 데이터베이스.
- Slack Webhook: 탐지 이벤트 발생 시 실시간 알림을 전달받는 외부 메시징 서비스.
- SMTP Server: 이메일 알림을 전송하기 위한 외부 메일 서버.
- Amazon S3 (Log Archive / Backup): 장기 로그 보관을 위한 외부 스토리지.

Interfaces

- SQL 트래픽: Client App ↔ DB-IDS ↔ MySQL (TCP 3306)
- 관리 인터페이스: Admin ↔ DB-IDS Dashboard (HTTPS REST/SSE)
- 알림 인터페이스: DB-IDS ↔ Slack (HTTPS Webhook), DB-IDS ↔ SMTP (TLS 587)
- 로그 아카이빙: DB-IDS ↔ Amazon S3 (HTTPS Upload)

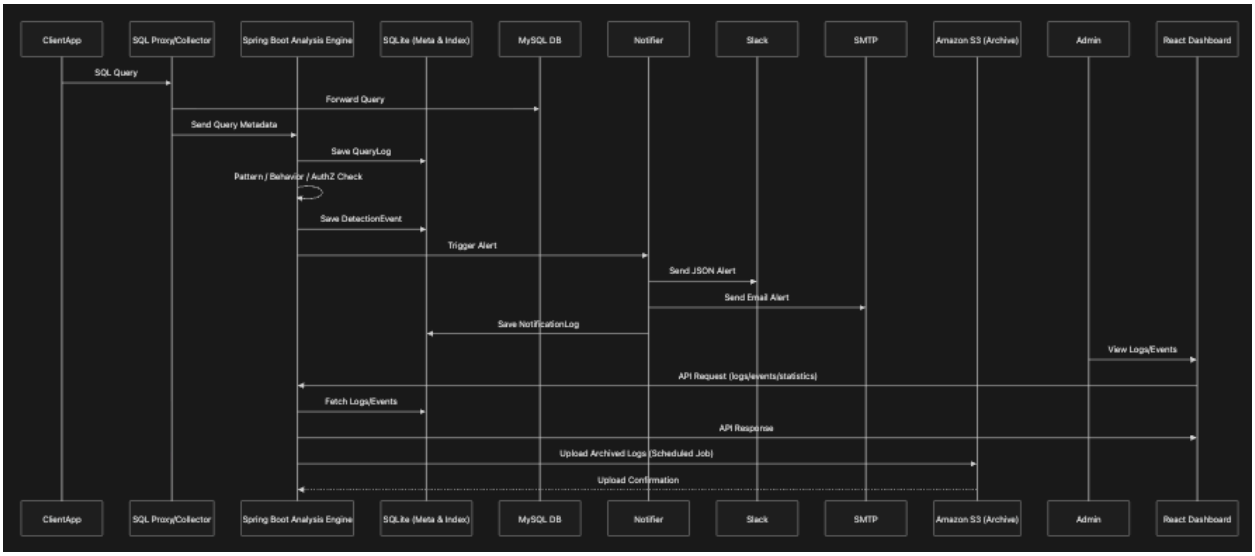
3.1.2. Container/Component View

이 뷰는 DB-IDS 시스템 내부 구조를 화이트박스로 열어보고, 주요 컴포넌트들의 역할을 간단하게 설명한다.

- SQL Proxy / Collector: Client Application의 SQL 요청을 가로채어 MySQL로 전달하고, 실행 메타데이터를 수집한다.
- Spring Boot Service: 핵심 분석 및 제어 컴포넌트로, 탐지 로직 수행, 이벤트 처리, 알림 전송, 로그 아카이빙 등을 담당한다.
- SQLite Meta & Index: 최근 로그 및 탐지 이벤트를 저장하는 메타스토어로, 빠른 조회를 지원한다.
- React Dashboard: 관리자가 로그와 이벤트를 조회·시각화하고, 설정을 제어할 수 있는 UI를 제공한다.
- Amazon S3 Log Archive: 장기 로그 아카이브를 담당하여 비용 효율적인 보관과 복구 기능을 제공한다.

3.1.3. Runtime/Interaction View (핵심 시퀀스)

본 절에서는 DB-IDS(Database Intrusion Detection System)가 실행 시 수행하는 주요 기능의 상호작용 과정을 설명한다. 본 뷰는 SRS에서 정의된 기능 요구사항을 기반으로 하며, 각 시나리오별로 시스템 내부 컴포넌트와 외부 액터 간의 동적 관계를 기술한다.



SQL 요청 처리 및 로깅 (FR-1)

1. Client Application은 SQL 쿼리를 DB에 전달한다.
2. SQL Proxy/Collector는 해당 요청을 가로채어 MySQL DB로 전달하면서, SQL 원문 및 실행 메타데이터(사용자, 실행 시간, 반환 행 수 등)를 추출한다.
3. 수집된 메타데이터는 Spring Boot Analysis Engine으로 전달된다.
4. Analysis Engine은 이를 SQLite 메타스토어에 QueryLog로 저장한다.

이상 탐지 (FR-2, FR-3, FR-4)

1. Analysis Engine은 수신한 쿼리를 정규화 및 파싱한 뒤, 탐지 엔진에 전달한다.
2. Pattern-based Engine은 금지된 SQL 시그니처(DROP TABLE, UNION SELECT 등)를 탐지한다.
3. Behavior-based Engine은 실행 시간, 반환 행 수, 쿼리 빈도를 정상 프로파일과 비교하여 이상 여부를 판단한다.
4. AuthZ Engine은 사용자 권한과 SQL 대상 객체를 비교하여 비인가 접근을 탐지한다.
5. 탐지된 이벤트는 DetectionEvent로 생성되어 SQLite에 기록된다.
6. 동시에 Event Router가 이벤트를 Notifier에 전달하여 알림을 트리거한다.

알림 전송 (FR-5)

1. Notifier는 DetectionEvent를 수신하면 알림 메시지를 구성한다.
2. 알림은 Slack Webhook API와 SMTP 서버를 통해 각각 실시간 메시지와 이메일로 전달된다.
3. 알림 성공/실패 여부는 NotificationLog로 기록된다.
4. 전송 실패 시 최대 3회 재시도를 수행하며, 실패 로그가 남는다.

관리자 대시보드 조회 (FR-6)

1. Administrator는 웹 브라우저를 통해 React Dashboard에 접근한다.
2. Dashboard는 Spring Boot API에 로그 및 이벤트 데이터를 요청한다.
3. API는 SQLite에서 데이터를 조회하고, 결과를 Dashboard에 반환한다.
4. Dashboard는 실시간 로그, 탐지 이벤트 목록, 통계 그래프를 시각화하여 관리자가 확인할 수 있도록 한다.

로그 백업 및 아카이브 (FR-7)

1. Scheduler/Jobs 모듈은 일정 주기마다 오래된 로그를 선별한다.
2. 로그 파일은 압축 처리 후 Amazon S3로 업로드된다.
3. 업로드 결과는 Analysis Engine과 SQLite에 기록되며, 실패 시 재시도된다.
4. 이를 통해 장기 로그 보관 및 규제 준수가 보장된다.

3.2. Decomposition Description (요소 상세 설명)

3.2.1. SQL Proxy / Collector

- 역할: Client Application의 모든 SQL 요청을 가로채고 MySQL로 전달한다.
- 책임: SQL 원문과 실행 메타데이터(사용자, 실행 시간, 반환 행 수 등)를 수집하여 Spring Boot Service로 전달한다.

3.2.2. Spring Boot Service (Analysis / Event Router / API SSE / Notifier / Jobs)

- Analysis Engine: Normalizer, Rule Engine, Behavior Engine, AuthZ Engine을 포함하며, 패턴 기반·행동 기반·권한 기반 탐지를 수행한다.
- Event Router: 탐지 결과를 종합하여 이벤트를 생성하고 저장 및 알림으로 분기한다.
- API / SSE: Dashboard에 로그 및 이벤트를 제공하며, 설정 관리 요청을 처리한다.
- Notifier: 탐지 이벤트를 Slack과 이메일로 전송하며, 실패 시 최대 3회까지 재시도한다.
- Jobs: 오래된 로그를 압축하여 Amazon S3로 아카이빙하고, 주기적 백업을 수행한다.

3.2.3. SQLite Meta & Index

- 역할: 최근 로그 인덱스, 탐지 이벤트, 관리자 계정, 알림 기록을 저장한다.
- 특징: Dashboard의 빠른 조회와 필터링을 지원하며, S3 아카이브의 원문 로그를 참조하는 메타데이터를 관리한다.

3.2.4. React Dashboard (UI)

- 역할: 관리자에게 실시간 이벤트 모니터링, 로그 조회, 통계 시각화 기능을 제공한다.
- 책임: 알림 채널과 탐지 규칙 설정을 관리하며, RBAC 기반 접근 제어를 적용한다.

3.2.5. Amazon S3 Log Archive

- 역할: 일정 기간이 지난 로그 파일을 장기 보관한다.
- 특징: 비용 효율적이고 내구성이 높은 스토리지로, 압축된 로그 파일을 보관하여 규제 준수와 장애 복구에 활용된다.

3.3. Design Rationale (설계 근거)

본 절에서는 3.1에서 제시된 아키텍처를 채택한 이유와 고려된 대안, 그리고 주요 트레이드오프를 설명한다.

3.3.1. Proxy 기반 아키텍처 선택

본 시스템은 SQL Proxy/Collector를 통해 DB 요청을 가로채고 로그를 수집하는 방식을 선택하였다. 이 접근법은 DB 서버 자체를 수정하거나 애플리케이션 레벨에 침입 로직을 삽입하지 않고도 모든 SQL 트래픽을 투명하게 감시할 수 있다는 장점이 있다.

- 이점: DBMS 독립성 보장, 유지보수 용이, 다양한 클라이언트 환경에 적용 가능
- 대안: DBMS 플러그인 방식
  - 배제 이유: 특정 벤더 종속성과 성능 저하 가능성, 클라우드 환경에서의 호환성 제약

3.3.2. Spring Boot 기반 분석 엔진

분석 엔진은 Spring Boot 프레임워크를 기반으로 구현되었다

- 이점: 안정적인 서버 개발 생태계, REST/SSE 지원 용이, 다양한 라이브러리 활용 가능
- 대안: Node.js 기반 로그 분석 모듈
  - 배제 이유: 실시간 이벤트 처리에는 적합하지만, 규칙 기반 및 통계 기반 탐지와 같은 연산 집중형 로직에서는 Java 기반 환경이 더 안정적이라고 판단하였다.

3.3.3. 이중 저장소 구조 (SQLite + S3)

로그 저장은 SQLite + Amazon S3의 이중 구조로 설계되었다.

- 이점:
  - SQLite: 경량 DB로 빠른 조회 및 Dashboard 응답 성능 보장
  - S3: 장기 보관, 아카이빙, 비용 효율성 확보
- 대안: 단일 RDBMS(MySQL)에 로그 및 이벤트를 모두 저장
  - 배제 이유: 로그 데이터 폭증 시 메인 DB 성능 저하 및 운영 리스크 증가

3.3.4. 단순 알림 채널 구조

알림 시스템은 Slack Webhook + SMTP 조합으로 설계되었다.

- 이점: 별도의 메시징 없이도 즉시 운영 가능, 관리 편의성
- 대안: 메시지 큐 기반 알림 처리.
    - 배제 이유: MVP 단계에서 인프라 복잡성을 증가시키고 운영 부담을 가중

3.3.5. 트레이드오프 요약

- 성능 vs 확장성: Proxy 방식은 소폭의 지연을 유발하지만, DB 독립성과 범용성을 위해 선택하였다.
- 단순성 vs 유연성: Slack/SMTP 직접 연동은 확장성은 낮지만, 초기 개발 및 운영 단순성을 위해 채택하였다.
- 단일 저장소 vs 이중 저장소: 단일 저장소는 관리가 단순하지만 성능 저하 위험이 크므로, 이중 저장소 구조로 설계하였다.

4. Data Design (데이터 설계)

4.1. Data Description (데이터 설명)

DB-IDS는 데이터베이스 접근 및 침입 탐지와 관련된 정보를 다양한 저장소에 관리한다. 데이터는 주로 실시간 조회용 단기 스토리지(SQLite)와 장기 보관용 아카이브(Amazon S3)로 구분되어 저장되며, 관리자는 Dashboard를 통해 이를 조회 및 분석할 수 있다. 주요 데이터 영역은 다음과 같다.

4.1.1. SQL Query Logs (쿼리 로그)

- 설명: Client Application이 실행한 모든 SQL 요청의 원문과 실행 메타데이터를 저장한다.
- 저장소: 단기 조회는 SQLite, 장기 보관은 Amazon S3에 압축 및 아카이브 형태로 저장된다.
- 포함 항목: LogID, UserID, 실행 시간, SQL 원문/요약, 반환 행 수, 실행 상태(성공/실패)
- 활용: 탐지 이벤트 추적, 관리자 감사, 성능 분석

4.1.2. Detection Events (탐지 이벤트)

- 설명: 패턴 기반, 행동 기반, 권한 기반 탐지 엔진에서 발생한 이상 행위 이벤트
- 저장소: SQLite
- 포함 항목: EventID, 관련 LogID, 탐지 유형, 심각도(Level), 발생 시간, 원본 쿼리
- 활용: 관리자 알림 트리거, 보안 감사 데이터

4.1.3.Notification Logs (알림 기록)

- 설명: Slack 또는 이메일로 전송된 보안 알림의 내역과 상태를 저장한다.
- 저장소: SQLite
- 포함 항목: NotifyID, EventID, 채널(Slack/Email), 전송 결과, 전송 시간
- 활용: 알림 성공률 모니터링, 장애 분석

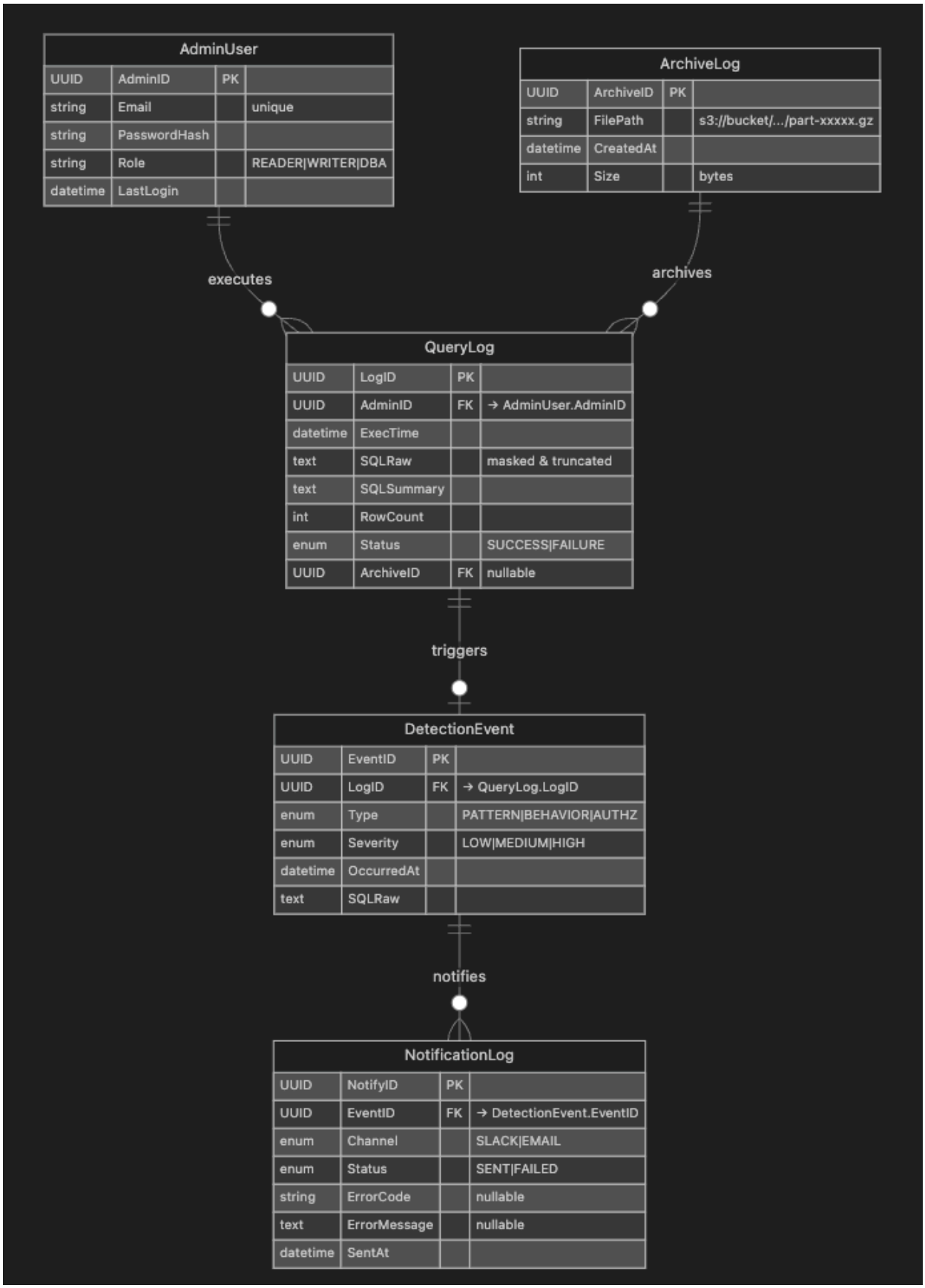
4.1.4. Admin Users (관리자 계정)

- 설명: Dashboard 접근 권한을 가진 관리자 정보와 인증 관련 데이터
- 저장소: SQLite
- 포함 항목: AdminID, PW 해시, 이메일, 최근 로그인 시간
- 활용: RBAC(Role-Based Access Control), 시스템 보안 관리

4.1.5. Archived Logs (아카이브 로그)

- 설명: 일정 기간(예: 30일) 이상 지난 SQL 쿼리 로그를 장기 보관하는 데이터
- 저장소: Amazon S3.
- 포함 항목: 압축된 로그 파일
- 활용: 보안 규제 준수, 장애 복구, 과거 분석

4.2. Data Dictionary (데이터 사전)


AdminUser
UUID AdminID PK
string Email unique
string PasswordHash
string Role READER|WRITER|DBA
datetime LastLogin
ArchiveLog
UUID ArchiveID PK
string FilePath s3://bucket/.../part-xxxxx.gz
datetime CreatedAt
int Size bytes
executes
archives
QueryLog
UUID LogID PK
UUID AdminID FK → AdminUser.AdminID
datetime ExecTime
text SQLRaw masked & truncated
text SQLSummary
int RowCount
enum Status SUCCESS|FAILURE
UUID ArchiveID FK nullable
triggers
DetectionEvent
UUID EventID PK
UUID LogID FK → QueryLog.LogID
enum Type PATTERN|BEHAVIOR|AUTHZ
enum Severity LOW|MEDIUM|HIGH
datetime OccurredAt
text SQLRaw
notifies
NotificationLog
UUID NotifyID PK
UUID EventID FK → DetectionEvent.EventID
enum Channel SLACK|EMAIL
enum Status SENT|FAILED
string ErrorCode nullable
text ErrorMessage nullable
datetime SentAt

4.2.1. QueryLog

Attribute	Type	Description
LogID	UUID	쿼리 로그 고유 식별자
AdminID	UUID (FK)	실행한 관리자 계정의 식별자 (→ AdminUser.AdminID)
ExecTime	Timestamp	실행 시각 (ISO 8601)
SQLRaw	Text	원본 SQL 쿼리
SQLSummary	Text	요약된 SQL 구문
RowCount	Integer	반환된 행(row) 수
Status	Enum	실행 상태 (SUCCESS / FAILURE)
ArchiveID	UUID (FK, nullable)	아카이브 파일 ID (→ ArchiveLog.ArchiveID)

4.2.2. DetectionEvent

Attribute	Type	Description
EventID	UUID	탐지 이벤트 고유 식별자
LogID	UUID (FK)	연관된 QueryLog의 LogID
Type	Enum	탐지 유형 (PATTERN / BEHAVIOR / AUTHZ)
Severity	Enum	심각도 (LOW / MEDIUM / HIGH)
OccurredAt	Timestamp	이벤트 발생 시각
SQLRaw	Text	탐지된 SQL 구문 (마스킹 적용)

4.2.3. NotificationLog

Attribute	Type	Description
NotifyID	UUID	알림 로그 고유 식별자
EventID	UUID (FK)	관련 탐지 이벤트 ID
Channel	Enum	알림 채널 (SLACK / EMAIL)
Status	Enum	전송 결과 (SENT / FAILED)
ErrorCode	String (nullable)	전송 실패 시 오류 코드
ErrorMessage	Text (nullable)	전송 실패 상세 메시지
SentAt	Timestamp	알림 전송 시각

4.2.4. AdminUser

Attribute	Type	Description
AdminID	UUID	관리자 계정 식별자
Email	String	관리자 이메일 주소 (unique)
PasswordHash	String	비밀번호 해시
Role	String	관리자 역할 (READER / WRITER / DBA)
LastLogin	Timestamp	최근 로그인 시각

4.2.5. ArchiveLog

Attribute	Type	Description
ArchiveID	UUID	아카이브 파일 고유 식별자
FilePath	String	S3 내 저장 경로 (예: s3://bucket/.../part-xxxxx.gz)
CreatedAt	Timestamp	아카이브 생성 시각
Size	Integer	파일 크기 (bytes)
RelatedLogIDs	JSON	포함된 QueryLog ID 목록

5. Compent Design (컴포넌트 설계)

본 장은 3.2에서 식별한 컴포넌트들의 핵심 동작을 의사코드로 기술한다. 각 절차는 입력/출력, 정상/오류 흐름을 포함한다.

5.1. SQL Proxy / Collector

5.1.1. handleConnection(clientSocket)

```
loop accept clientSocket:
  dbSocket := connectTo(MySQL)
  spawn pipe(clientSocket → dbSocket, onRequest)
  spawn pipe(dbSocket → clientSocket, onResponse)

procedure onRequest(sqlPacket):
  ts := now()
```

```
  forward(sqlPacket) to dbSocket
  recordContext.currentQueryStart := ts
  recordContext.user := extractUser(sqlPacket)
  recordContext.sqlRaw := extractSQL(sqlPacket)

procedure onResponse(resultPacket):
  elapsed := now() - recordContext.currentQueryStart
  rows := extractRowCount(resultPacket)
  status := extractStatus(resultPacket)
  meta := {
    user: recordContext.user,
    sql: recordContext.sqlRaw,
    execTime: elapsed,
    rows: rows,
    status: status,
    timestamp: now()
  }
  enqueueAsync(meta) to AnalysisService
  forward(resultPacket) to clientSocket
```

5.2. Analysis Engine (Spring Boot)

5.2.1. `processQueryMeta(meta)`

```
norm := Normalizer.normalize(meta.sql)
features := Parser.extract(norm)

ruleVerdict := RuleEngine.evaluate(features)
behVerdict  := BehaviorEngine.score(meta.user, features, meta.execTime, met
a.rows)
authVerdict := AuthZEngine.check(meta.user, features)

event := EventRouter.route(meta, norm, ruleVerdict, behVerdict, authVerdict)
Storage.saveQueryLog(meta, norm.summary)
```

```
if event != null:
  Storage.saveDetectionEvent(event)
  Notifier.trigger(event)
```

5.2.2. Normalizer.normalize(sql)

```
sql1 := trimWhitespace(sql)
sql2 := uppercaseKeywords(sql1)
sql3 := maskLiterals(sql2)
summary := summarize(sql3)
return { normalized: sql3, summary }
```

5.2.3. RuleEngine.evaluate(features)

```
for rule in RuleSet.active:
  if rule.pattern.matches(features):
     return {match: true, ruleId: rule.id, severity: rule.severity}
return {match: false}
```

5.2.4. BehaviorEngine.score(user, features, execTime, rows)

```
profile := Baseline.get(user, features.tables, currentHour)
freqΔ   := zscore( profile.freq,     currentFreq(user) )
rowsΔ   := zscore( profile.rowcount, rows )
latΔ    := zscore( profile.latency,  execTime )

score := 0.4*freqΔ + 0.3*rowsΔ + 0.3*latΔ
flag  := (score >= 0.8)

return {score: clamp(score,0,1), flag: flag}
```

5.2.5. AuthZEngine.check(user, features)

```
role := RBAC.roleOf(user)
for obj in features.objects:
  if not RBAC.isAllowed(role, features.verb, obj):
    return {violation: true, policyId: findPolicy(role, features.verb, obj)}
return {violation: false}
```

5.2.6. EventRouter.route(meta, norm, ruleVerdict, behVerdict, authVerdict)

```
if ruleVerdict.match or behVerdict.flag or authVerdict.violation:
  sev := maxSeverity(ruleVerdict.severity, scoreToSeverity(behVerdict.score),
violationSeverity)
  return {
    eventId: uuid(),
    logId: meta.logId,
    type: classify(ruleVerdict, behVerdict, authVerdict),
    severity: sev,
    occurredAt: now(),
    sqlRaw: meta.sql
  }
return null
```

5.3. API / SSE (Spring Boot)

5.3.1. GET /api/query-logs

```
validateAuth()
params := {from, to, user, table, status, page, size}
return Storage.findQueryLogs(params)
```

5.3.2. GET /api/events

```
validateAuth()
params := {from, to, type, severity, page, size}
```

```
return Storage.findEvents(params)
```

5.3.3. `GET /api/events/stream` (SSE)

```
validateAuth()
openSSE()
subscribe(EventBus)
onEvent(e): sse.send(e)
```

5.4. Notifier

5.4.1. `trigger(event)`

```
msg := template(event)
results := []
results += trySendSlack(msg)
results += trySendEmail(msg)
Storage.saveNotificationLog(event.id, results)
```

```
function trySendSlack(msg):
  retry up to 3 with backoff:
    resp := http.post(slackWebhookURL, json=msg)
    if resp.ok: return {channel:"SLACK", status:"SENT", ts:now()}
  return {channel:"SLACK", status:"FAILED", ts:now()}

function trySendEmail(msg):
  retry up to 3 with backoff:
    resp := smtp.send(to=adminList, subject=msg.title, body=msg.body)
    if resp.ok: return {channel:"EMAIL", status:"SENT", ts:now()}
  return {channel:"EMAIL", status:"FAILED", ts:now()}
```

5.5. Storage (SQLite + S3)

5.5.1. `saveQueryLog(meta, summary)`

```
insert into QueryLog(LogID, UserID, ExecTime, SQLRaw, SQLSummary, RowC
ount, Status)
values(uuid(), meta.user, meta.timestamp, maybeTruncate(meta.sql), summar
y, meta.rows, meta.status)
```

5.5.2. `saveDetectionEvent(event)`

```
insert into DetectionEvent(...) values(event...)
```

5.5.3. `archiveJob()` (Scheduler/Jobs)

```
candidates := QueryLog.where(ExecTime < now()-days(KEEP_DAYS) and Arc
hiveID is null)
batch := makeFile(candidates)
s3key := s3.upload(batch.file)
archId := insert ArchiveLog(ArchiveID=uuid(), FilePath=s3key, CreatedAt=now
(), Size=batch.size)
update QueryLog set ArchiveID=archId where LogID in batch.logIds
delete or compact local raw bodies if policy says so
```

5.6. Dashboard (React)

5.6.1. `useLogsQuery(filters)`

```
call GET /api/query-logs with filters & pagination
render Table with virtualized rows
```

5.6.2. `useEventStream()`

```
evtSrc := new EventSource('/api/events/stream', withAuthToken)
onmessage(e) ⇒ push to realtime table & toast
```

6. Requirements Matrix (요구사항 매트릭스)

이 절은 SRS에서 정의된 기능 요구사항과 본 설계 문서의 컴포넌트 및 데이터 구조 간의 추적 관계를 나타낸다. 각 요구사항은 관련 컴포넌트 및 데이터 엔티티와 매핑되며, 이를 통해 요구사항 충족 여부를 검증할 수 있다.

Requirement ID	Requirement Description	Related Components	Related Data Entities
FR-1	모든 SQL 쿼리를 로깅한다	SQL Proxy / Collector, Spring Boot API	QueryLog
FR-2	패턴 기반 공격(SQL Injection, DROP TABLE, UNION SELECT 등)을 탐지한다	Spring Boot Analysis Engine (Rule Engine)	DetectionEvent
FR-3	행동 기반 탐지(쿼리 빈도, 반환 행 수, 실행 시간)를 수행한다	Spring Boot Analysis Engine (Behavior Engine)	QueryLog, DetectionEvent
FR-4	권한 기반 탐지(비인가 계정의 시스템 테이블 접근 등)를 수행한다	Spring Boot Analysis Engine (AuthZ Engine)	AdminUser, QueryLog
FR-5	탐지 이벤트 발생 시 관리자에게 알림을 전송한다	Notifier (Slack, Email)	NotificationLog
FR-6	관리자가 실시간 이벤트 및 통계를 확인할 수 있다	React Dashboard, Spring Boot API (SSE)	QueryLog, DetectionEvent
FR-7	로그를 장기 보관하여 추후 분석 및 규제 준수에 활용한다	Jobs (S3 Upload), Amazon S3	ArchiveLog