

모바일 엔드포인트 행동 분석 기반
Policy Engine 구현



팀명: KGL

지도교수: 최윤희

202255655 권태현

202255654 구현서

202255668 이승원

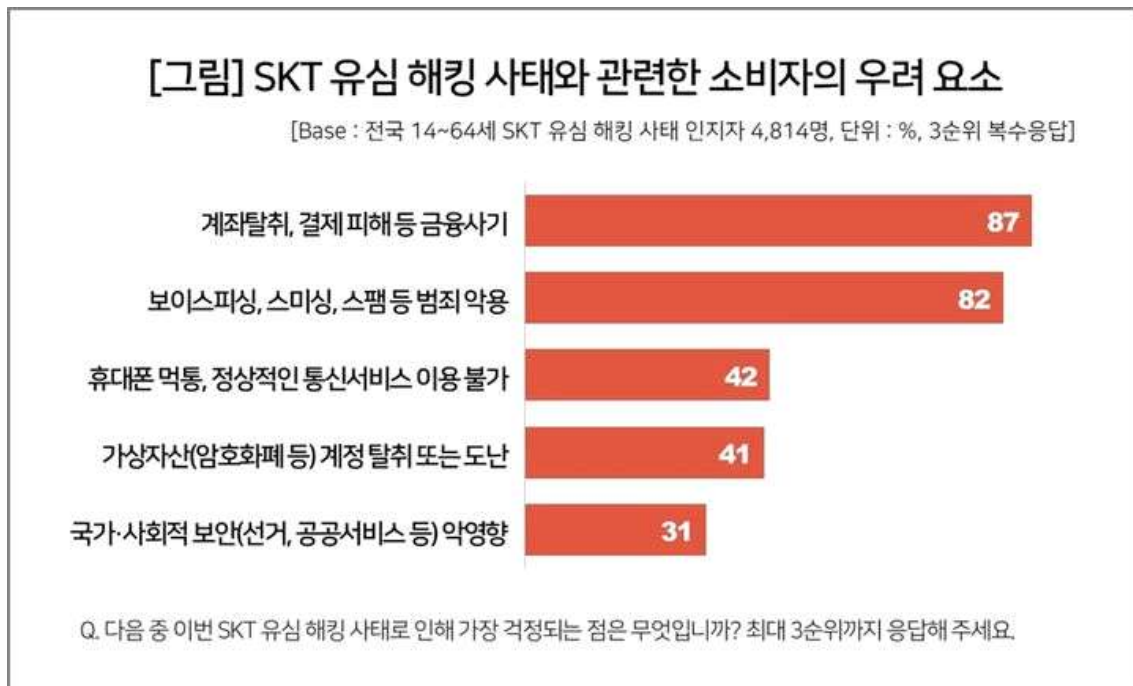
목차

1. 과제 배경 및 연구 목적
 - 1.1 과제 배경
 - 1.2 연구 목적
2. 요구 조건 및 제약 사항
 - 2.1 요구 조건
 - 2.2 유스케이스
 - 2.2 제약 사항
3. 과제 구체화 방안
 - 3.1 전체 구조도
 - 3.2 백엔드
 - 3.3 프론트엔드
 - 3.4 앱 어플리케이션
 - 3.5 머신러닝 모델
4. 수행 현황
 - 4.1 백엔드
 - 4.2 프론트엔드
 - 4.3 앱 어플리케이션
 - 4.4 머신러닝 모델
5. 구성원별 진척도
6. 향후 추진 계획

1. 과제 배경 및 연구 목적

1.1 과제 배경

오늘날 클라우드 서비스와 모바일 환경의 확산으로 사용자 인증과 보안의 중요성이 크게 증가하고 있다. 특히 스마트폰과 같은 모바일 기기는 민감한 개인 데이터를 갖고 있기 때문에 더욱 보안 공격에 예민하게 반응해야 한다. 최근에는 SK텔레콤의 해킹피해로 수천만명의 개인정보가 위협받기도 했다. 많은 이들이 이러한 보안 피해에 불편함을 겪었고, 이는 모바일 보안이 얼마나 중요한 문제인지 여실히 보여주는 큰 사례가 되었다.



기존의 정적인 경계 기반 보안 및 방화벽은 내부자 위협이나 계정 탈취와 같은 사용자 기반 공격에 취약하다는 점이 꾸준히 지적되어 왔다. 최근에는 사용자 행동 기반 이상 탐지(UEBA)를 통해 정상적인 활동 패턴을 학습하고, 이를 벗어나는 비정상 행위를 실시간으로 탐지하려는 시도가 활발해졌다. 그러나 현실에서는 명확히 라벨링된 공격 데이터를 수집하기 어렵기 때문에, 비지도 학습 기반의 유연한 이상 탐지 시스템이 요구된다.

1.2 연구 목적

본 연구는 모바일 단말 사용자의 행위 데이터를 활용하여 비지도 학습 기반 이상 탐지 모델을 개발하고 이를 네트워크 접근 제어와 연계하는 NAC Agent를 구현하는 것을 목표로 한다. 이를 위해 실제 사용자 로그를 바탕으로 정상 행위 프로필을 구축하고, 벗어나는 패턴을 실시간으로 탐지할 계획이다. 탐지된 이상 징후는 모바일 단말의 동적 접근 통제를 위해 정책 엔진과 연동되어 네트워크 접근 권한을 즉시 조정한다. 또한 네트워크 접근이나 위치정보등 모바일 기기 데이터를 활용하여 사용자 인증 신뢰성을 보완할 수 있도록 설계할 예정이다. 궁극적으로는 기존 정적 인증 체계를 보완하는 지능형 NAC 프로토타입을 완성하여 동적인 사용자 환경에서도 실질적인 보안 효과가 있는 시스템을 구현하고자 한다.

2. 요구 조건 및 제약 사항

2.1 요구 조건

1)엔드포인트 에이전트

- 모바일 사용자 단말에서 위치·터치이벤트등의 로그 데이터를 실시간으로 수집한다.
- 수집한 데이터를 로컬 DB에 저장하고, 서버 API를 통해 전송한다.
- 단말 리소스 소모를 최소화하기 위해 핵심 속성만 전처리·벡터화하여 전송한다.

2)로그 콜렉터 및 로그 수집 서버 구축

- Django REST API 등을 이용하여 모바일 단말로부터 수집된 로그 데이터를 수신한다.
- 수신된 데이터를 SQLite 등을 이용하여 DB에 저장하고, 실시간으로 UEBA 분석 및 정책 엔진에 반영한다.

3)행동 분석 및 이상 탐지 모듈 구현

- Isolation Forest, LSTM Auto-Encoder 등 비지도 학습 기반 모델을 통해 정상 행동 패턴을 학습한다.
- 요청 속성 등을 입력으로 사용, 재구성 오차 기반으로 이상 여부를 탐지한다.
- 탐지된 이상 징후를 정책 엔진과 대시보드로 전달한다.

4)정책 엔진 개발

- 로그, 위치, 시간대 등의 속성을 기준으로 정적 규칙 기반 필터링을 수행한다.
- 이상 탐지 결과와 결합하여 실시간으로 네트워크 접근 제어 여부를 평가하고, 이상 현상 발생시 네트워크 접근을 차단하거나 추가 인증을 사용자에게 요구한다.
- 정책 룰이나 threshold는 관리자가 직접 추가·수정할 수 있다.

5)관리자 대시보드

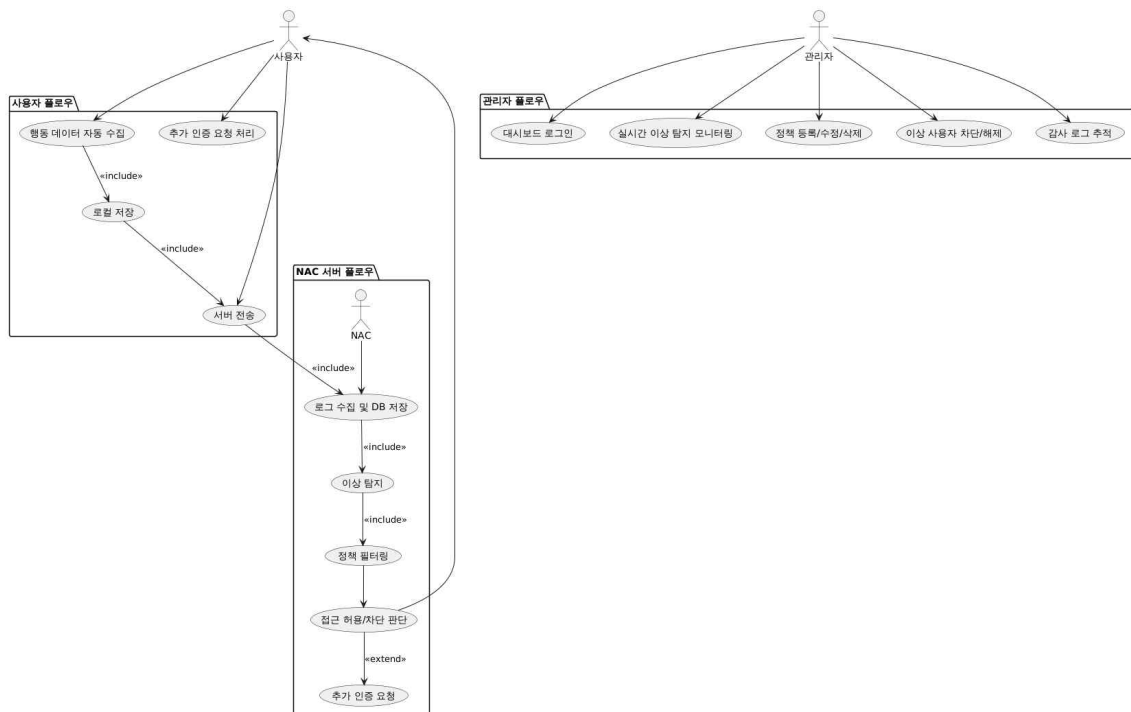
- 이상 탐지 결과를 실시간으로 시각화하여 관리자에게 제공한다.

-사용자별 행동 이력, 접근 시도 현황, 정책 위반 내역 등을 한눈에 확인할 수 있다.

6)시스템 아키텍처 및 개발 환경

- ZTNA(Zero Trust Network Access) 구조를 반영하여 설계한다.
- 로컬/클라우드 연동으로 동작하며 UEBA 학습에는 공개 데이터셋 활용.
- 모듈 간 API 설계는 RESTful 구조를 기본으로 하고, 로그 전송과 정책 처리 흐름이 실시간으로 연동되도록 한다.

2.2 유스케이스



이 유스케이스는 NAC 시스템에서 사용자와 관리자가 수행할 수 있는 주요 동작 흐름을 보여준다. 사용자는 자신의 행동 데이터를 서버에 보내고, 서버의 추가 인증 요청을 처리한다. 사용자는 이상 행동을 탐지하는 NAC 서버의 제어를 받게 된다. 관리자는 대시보드 모니터링을 통해 실시간으로 이상현상을 모니터링할 수 있으며, 정책 등록/변경/삭제, 사용자 제어 이력 확인 등의 작업을 수행할 수 있다. NAC 서버는 필터링 모듈과 제어 모듈을 포함하며, 로그 기록, 이상 탐지 결과 알림, 정책 적용, 접근 제어 명령 전달 등의 기능을 제공한다.

전체적으로 이 시스템은 사용자 단말에서 수집된 행동 및 로그 데이터 기반으로 이상 탐지를 수행하고, 실시간으로 네트워크 접근 권한을 동적으로 제어하는 구조이다.

2.3 제약 사항

2.3.1)Time-slice 기반 사용자 데이터 분할 처리

로그 및 행동 데이터 수집 시 큰 Time-slice(30초)를 설정하여 분할처리 하는 방식은 연산 부담이 최소화 된다는 장점이 있지만, 실시간 정책반영에서 사용자 UX가 떨어진다는 느낌을 줄 수 있다. 이러한 단점을 바탕으로 agent 구현시 실시간 정책 반영 방법을 구상해야 할 것이다.

-해결 방안-

위에서 언급한 단점 때문에 우리는 사용자 데이터를 더욱 짧은 time-slice로 REST 서버에 전송할 계획이다. 사용자가 소규모라면 실시간으로 데이터를 전송하여도 서버 측 부하가 적을 것이라 판단했기 때문이다. 또한 이미 NAC agent가 사용자 로그 및 행동 데이터를 전처리하여 서버에 보내기 때문에 연산 부담은 고려대상이 아니라 판단했다.

위에서 언급한 단점에도 불구하고 Time-slice 기법은 서버측 부하를 줄이는 데에 쓸만한 방법이라고 생각한다. 그래서 더욱 큰 Time-slice를 사용하는 기법은 사용자가 많아질 때에 사용하기 적합한 방법일 것이다.

2.3.2)CLUE-LDS 오픈소스 데이터

CLUE-LDS는 우리가 사용할 모바일 환경의 행동 속성과 일치하지 않는 속성이 많았다. 따라서 UEBA 모델 학습 시 모바일 사용자의 실제 이상 징후를 정확히 판단하기 어렵다고 판단했다.

-해결 방안-

CLUE-LDS 데이터셋을 머신러닝 모델의 개발 및 성능향상 목적으로만 사용, 그렇게 보완된 모델을 바탕으로 실제 사용자의 로그를 Input으로 입력 받으면 시계열로 분석하는 UEBA 모델을 개발할 예정이다.

CLUE-LDS가 아닌 다른 데이터셋이 우리가 개발할 모델과 잘 맞는다면 그 모델을 채용하는 것도 방법일 것이다.

2.3.3)다른 앱에서의 터치 이벤트

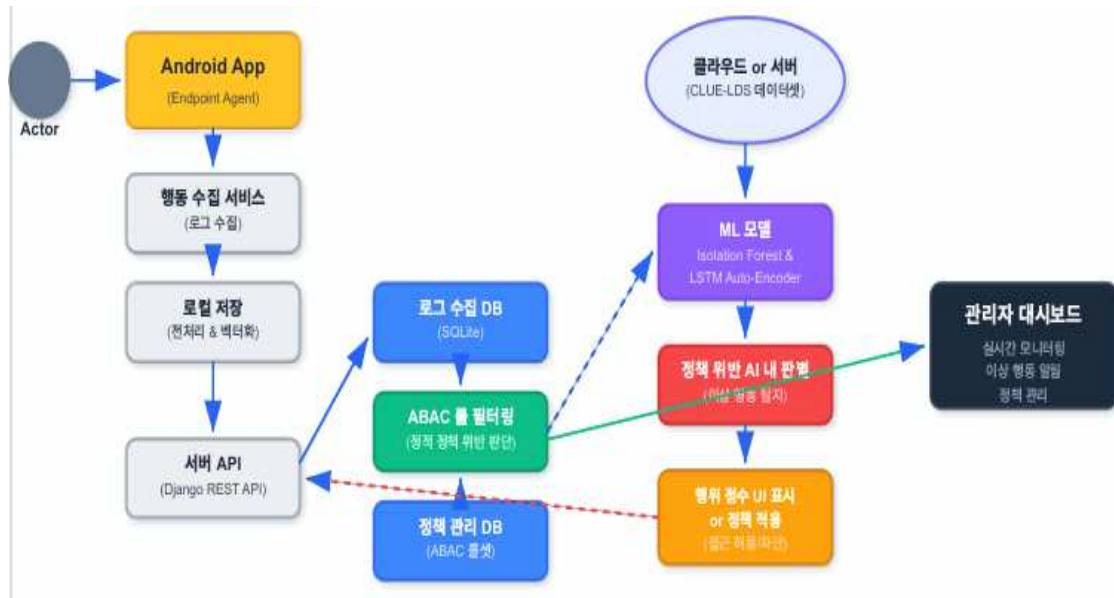
Android 보안 정책에 의해 다른 앱의 터치 이벤트에 접근할 수 없음. Accessibility Service를 통한 접근은 사용자의 명시적 권한 증인이 필요하며, Google Play Store 정책상 제한됨.

-해결 방안-

자체 앱 내 터치 패턴 상세 분석으로 대체. Policy Engine 앱 내에서만 터치 이벤트를 수집하여 이상패턴을 감지한다. 터치 압력, 속도, 리듬, 접촉 면적 등 상세한 터치 분석을 통해 추가 인증 필요여부를 판단한다.

3. 과제 구체화 방안

3.1 전체 구조도



1) 모바일 클라이언트

- 사용자의 행동 수집 서비스는 위치 정보등의 로그 기록 등을 수집한다.
- 수집된 로그는 전처리를 거쳐 로컬 저장한 뒤 서버 API를 통해 전송된다.
- 클라이언트에서 ABAC 속성에 필요한 핵심 값만 추출·전처리하여 데이터 전송 비용과 디바이스 리소스를 최소화한다.
- 관리자가 아닌 일반 사용자는 자신의 로그 기록만을 볼 수 있으며, 행동 및 로그 기록을 바탕으로 네트워크 접근이 차단될 수 있다.

2) 로그 수집 서버(서버 API & 로그 수집 DB)

- Django REST API를 통해 모바일 클라이언트에서 전송된 데이터를 수신한다.
- 수집된 로그 데이터는 SQLite 등 DB에 저장되어 ML 모델 학습/탐지 모듈과 연계된다.
- 실시간으로 정책 필터링 엔진과 연결되어 요청 흐름을 관리한다.

3) 행동 분석 및 이상 탐지 모듈(ML 모델)

- Isolation Forest와 LSTM Auto-Encoder를 사용하여 정상 행동 패턴을 학습한다.
- 사용자 속성, 요청 속성등을 입력으로 사용하여 재구성 오차 기반으로 이상을 탐지한다.
- 탐지된 로그 및 행동패턴은 정책 엔진으로 전달되어 동적 접근 제어에 활용된다.

4) 정책 관리 DB(ABAC 룰 매칭)

- 사용자, 위치, 로그, 시간대 등의 속성을 기반으로 정적 접근 통제를 먼저 수행한다.
- 이상 탐지 모듈의 출력 결과와 결합하여 정책 위반 시 알림 및 즉시 접근 차단 여부를 판단한다.
- 룰은 정책 관련 DB에 저장된다.

5) 관리자 인터페이스

- 관리자 대시보드는 실시간 모니터링과 탐지 결과 시각화를 제공한다.
- 이상 사용자 활동 이력, 정책 위반 상황, 정책 수정 기능을 지원한다.
- 최종적으로 관리자 판단에 따라 수동 차단 및 정책 관리가 가능하다.

3.2 백엔드

백엔드 서버는 Python 기반 Django REST framework를 사용하여 모바일 클라이언트로부터 사용자 행동 데이터를 수신하고 구조화된 데이터베이스에 저장한다.

저장된 데이터는 TensorFlow로 학습된 이상 탐지 모델과 연동되어 실시간으로 정상 및 이상 여부를 판단하며, ABAC 정책 엔진과 결합하여 최종 접근 허용 여부를 결정한다.

또한 관리자용 대시보드가 REST API를 통해 감지된 이상 행동, 차단 이력, 정책 관리를 손쉽게 할 수 있도록 직관적인 API를 제공하고, Postman과 Swagger를 활용하여 API 품질과 문서화를 유지한다.

백엔드 설계 사항

- ① 데이터 수신 & 저장 - 클라이언트로부터 행동 로그 받기 → DB에 저장
- ② 머신러닝 모델 연동 - 저장된 로그를 기반으로 ML 모델에 요청 송신
- ③ 정책 엔진 처리 - ABAC 룰셋 + ML 결과로 접근 허용 여부 판단
- ④ API 설계 & 대시보드 연동 - 관리자 화면이 REST로 서버에서 결과를 받아볼 수 있도록 제공

1) 데이터 수신 & 저장 모듈

목표:

- 모바일 앱에서 사용자 행동 데이터(위치, 요청 로그) 수신
- JSON 형태의 요청을 파싱하고 Django ORM을 통해 DB에 구조적으로 저장
- 수신 과정에서 속성 벡터(사용자, 요청, 통계 속성)를 생성하거나, 클라이언트가 벡터화한 것을 그대로 받음

DB설계:

- UserBehaviorLog 테이블: user_id, device_id, request_time, request_url, request_method, stat_feature_json 등
- PolicyRule 테이블: ABAC 정책 룰셋 저장 (역할, 위치, 시간대 등)
- AnomalyResult 테이블: 모델 추론 결과 저장 (score, is_anomaly flag 등)

툴:

Django + DRF, SQLite (초기), PostgreSQL (확장)

2) 머신러닝 모델 연동

목표:

- 저장된 로그 데이터 or 실시간 속성 벡터를 비동기 or 동기 ML 모듈에 전달
- LSTM AE, Isolation Forest 모델을 Django에서 직접 로드하여 추론

구현 아이디어:

- 서버 시작 시 모델 파일 로드 (lazy loading)
- Django View에서 numpy로 벡터화 → model.predict() 호출 → 오차 점수 계산 → 이상 여부 판단
- 추후 대규모라면 Celery + Redis로 비동기 큐 구성 고려가능

툴:

- TensorFlow, Numpy, Scikit-learn
- Jupyter Notebook: 추론 로직은 사전에 검증 후 서버에 옮기기

API 흐름:

- 클라이언트 → 서버(로그 저장) → 저장된 로그 → 모델 추론 → 결과 DB 기록 → 접근 허용 여부 판단 → 응답

3) 정책 엔진 모듈

목표:

- ABAC 룰셋 + ML 결과 결합하여 접근 제어 로직 완성
- 예: 정상 행동이지만 정적 정책 위반(야간접속, 특정 IP 불가)이면 차단

구현 아이디어:

- PolicyEngine 모델에 위치, 시간대 조건 등 정의
- View 단에서 ML 이상 점수와 ABAC 룰 결과를 조합 → 최종 판단
- PolicyEngine 클래스로 모듈화하여 재사용성 확보

툴:

- Django ORM 로직 + rule 테이블 매칭
- rule 기반으로 필터링 후 이상 탐지 결과 반영

4) API 설계 및 대시보드 연동

목표:

- 관리자용 대시보드가 REST API를 통해 감지된 이상 행동, 접근 차단 현황, 정책 편집 등을 할 수 있도록 지원
- Postman, Swagger로 API 테스트 및 문서화

구현 아이디어:

- /api/logs/ : 저장된 행동 로그 리스트/필터링
- /api/anomalies/ : 이상 탐지 결과 목록 제공
- /api/policies/ : 정책 룰 CRUD
- /api/access-control/ : 실시간 접근 시도에 대한 허용/차단 여부 응답
- /api/dashboard/ : 주요 지표(최근 이상 탐지 건수, 사용자별 통계 등)

툴:

- Django REST framework (ViewSet, Router, Permission)
- Swagger
- Postman -> 테스트 케이스 작성

3.3 프론트엔드

1) 주요 목표

- Django REST API와 연동하여 실시간 이상 탐지 결과, 정책 룰 관리, 로그 모니터링 등을 관리자 대시보드에서 직관적으로 시각화한다.
- 이상 탐지 현황, 사용자별 통계, 정책 상태 등을 카드, 차트 등으로 시각화하여 즉각적인 모니터링과 정책 수정이 가능하도록 한다.

2) 페이지 구성

- 로그인 페이지: 관리자나 사용자가 로그인 할 수 있는 페이지. 사용자와 관리자는 이 단계에서 인증과정을 거치게 된다.
- 메인 대시보드: 최근 이상 탐지 건수, 사용자 이상 비율 등을 카드 & 차트로 표시. 접근 시도/차단 현황을 테이블로 구성하여 시각화. 최근 n개의 이상 수치를 그래프화 함.
- 로그 관리 페이지: 서버 API를 이용하여 사용자의 로그를 로드함. 차단 여부 및 이상 수치 표시됨.
- 정책 관리 페이지: ABAC 룰 등록/수정/삭제할 수 있는 페이지.

3)상태 관리 및 데이터 흐름

상태관리:

- Redux Toolkit 또는 React Query로 REST 데이터 요청/캐싱
- 사용자 인증 토큰 상태 (Context API 또는 Redux)

컴포넌트 구조:

- 각 페이지는 /pages에 라우팅, 공통 컴포넌트는 /components로 모듈화
- chart.js 등의 시각화 라이브러리 활용
- Tailwind CSS를 이용한 반응형 UI

형상 관리:

- Git, Github를 이용한 프론트엔드 프로젝트 형상 관리
- Github 주소: https://github.com/bootaeng/KGL_NAC

3.4 앱 어플리케이션

1)데이터 수집 및 전송 최적화

현재 주기적으로 쌓인 데이터를 서버에 전송하고 안드로이드 에뮬레이터를 이용 중이기에, 추후 실제 스마트폰을 사용한 데이터 수집 결과를 확인한 후 전송 주기, 데이터 형식, 데이터 개수, 사용하지 않는 데이터 조절 및 제거 예정

세부 개선 계획:

- 적응형 학습 단계 구현: 초기 몇십분(예:30~60분)동안 집중적인 데이터 수집을 통해 사용자의 기본 사용 습관 및 패턴을 머신러닝으로 학습
- 학습 완료 후 모드 전환: 이후 데이터는 이상반응 감지, 잘못된 접근, 비정상 패턴 탐지용으로 활용하여 수집 빈도 및 데이터량 최적화
- 실기기 테스트 기반 최적화: 서버 과부하 및 배터리 사용량을 실제 스마트폰에서 측정하여 모델 학습 및 이상 반응 감지 목적에 최적화된 데이터 전처리, 선별적 수집, 불필요 데이터 제거 적용

추가 고려 사항:

2단계 수집 전략

- phase 1(학습 단계): 고빈도 수집(15초 주기, 전체 데이터)
- phase 2(감지 단계): 저빈도 수집(60초 주기, 핵심 데이터만)
- 배터리 임계값 연동: 배터리20% 이하시 최소 모드 전환
- 네트워크 최적화: WIFI/셀룰러별 차등 데이터 전송

2)사용자 편의성 및 UI/UX 개선

머신러닝을 통해 사용자가 이상 반응 행동을 보였을 때 앱에서 이를 보고하는 방식을 구현하며, 실제 스마트폰 테스트를 통해 더욱 사용자 친화적인 앱으로 변모하기 위한 UI/UX 변경 예정

세부 개선 계획:

- 단계별 알림 시스템: 이상 탐지 수준에 따른 차등 알림

-설명 가능한 알림: 단순한 이상 감지 메시지가 아닌, 구체적인 행동 패턴 변화 설명 제공

-사용자 피드백 연동: 잘못된 탐지에 대한 신고 기능으로 모델 개선

추가 UI/UX 개선 사항:

-학습 진행률 표시: “사용 패턴 학습 중 ... 85% 완료”

-투명성 강화: 수집되는 데이터 종류 및 용도 실시간 표시

-프라이버시 컨트롤: 세부 기능별 on/off 설정

-접근성 고려: 시각/청각 장애인을 위한 대체 알람 방식

3.3 머신러닝 모델

1)주요 목표

모바일 클라이언트에서 수집된 BehaviorLog(위치, 요청, 디바이스 정보)를 속성 벡터로 변환해 실시간으로 이상 여부를 판단한다. 정상 데이터를 기반으로 비지도 학습(Unsupervised)하여 정상 행동 패턴을 학습하고, 정상 행동 패턴에서 많이 벗어나는 행동은 이상으로 탐지한다. Isolation Forest와 LSTM Auto-Encoder를 병행하여 단일 로그 + 시퀀스 기반 이상 탐지를 결합, 탐지 신뢰도를 고양한다. 최종 탐지 결과는 ABAC 정책 엔진과 결합해 접근 허용/차단/추가 인증 여부를 결정하게 된다.

2)입력 데이터 및 전처리

입력 데이터:

-사용자 속성: user_id, device_info(model, os_version, network_type 등)

-사용자 로그: 위치, 터치등의 사용자 행동 데이터

전처리 흐름:

-Django ORM 으로 최근 로그 n건 조회

-범주형 속성과 수치형 속성을 인코딩 or 스케일링

-Isolation Forest: 1건 단위 벡터

-LSTM AE: 시계열 시퀀스 배열 (ex: 최근 10~30건)

3) 판단 로직

-Isolation Forest

입력: 단일 행동 벡터

출력: 이상 점수 (0.0~1.0)

판단: 이상 점수 > threshold(예: 0.6)이면 이상 행동

-LSTM Auto-Encoder

입력: 시계열 행동 시퀀스 (벡터 리스트)

출력: 재구성 오차 (Reconstruction Error)

판단: 오차 > threshold(예: 0.05)이면 이상 행동

-최종 판단

두 모델의 결과를 종합해 신뢰도 or 이상 수치 생성

ABAC 정책 조건(위치/시간대/역할)과 조합하여 PolicyResponse 생성 -> 정책 반영하여 네트워크 접근 차단 등의 agent 행동 수행

4) 백엔드 연동 방식

Django 백엔드 연동

-서버 시작 시 머신러닝 모델 파일 로드 → 메모리에 상주

-요청 시 View 함수가 벡터를 생성하여 model.predict() 호출

-추론 결과는 AnomalyResult 테이블에 저장

-최종 결과는 PolicyEngine 모듈과 결합하여 allow / block / warning / additional_auth 로 반환

4. 수행 현황

4.1 백엔드

1) Behavior Log API POST 테스트 화면

The screenshot shows the Swagger UI for the 'Behavior Log List' endpoint. The breadcrumb is 'Api Root / Behavior Log List'. The endpoint is 'POST /api/behavior-logs/'. The response is shown as a JSON object: `{ "status": "success", "decision": "allow" }`. Below the response, there is a form to test the endpoint. The 'Media type' is set to 'application/json'. The 'Content' field is empty. A 'POST' button is at the bottom right.

Django REST Framework 기본 Swagger 형태로 구현된 `/api/behavior-logs/` 엔드포인트가 POST 요청을 수신하고, 정상적으로 행동 로그와 이상탐지 결과를 생성하여 Stub 응답 (status, decision)을 반환함을 확인하였다.

2) Anomaly 결과 DB 저장 현황

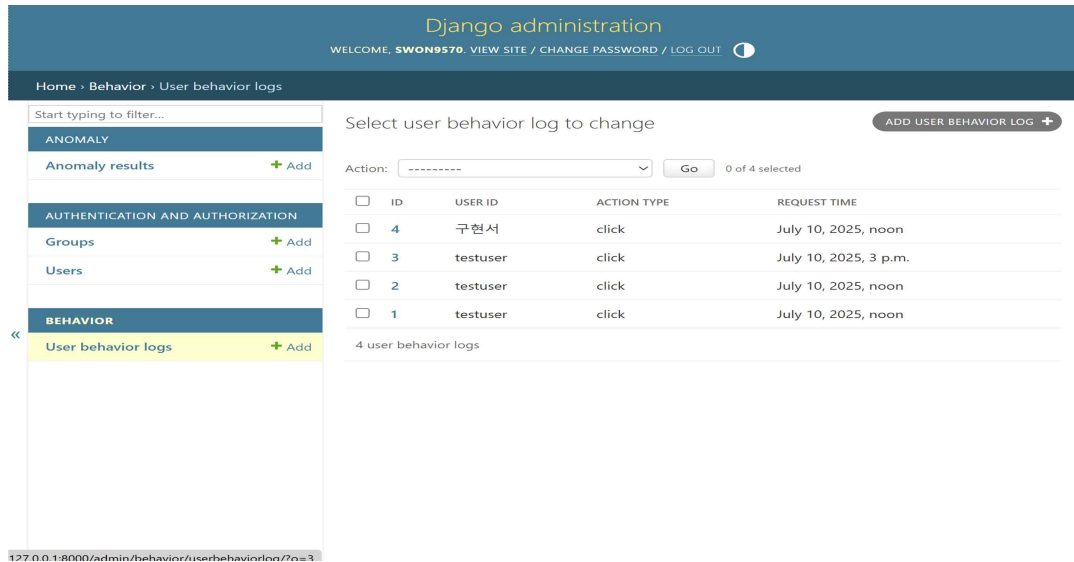
The screenshot shows the Django administration interface for 'Anomaly results'. The left sidebar has a search bar and a menu with 'ANOMALY', 'AUTHENTICATION AND AUTHORIZATION', and 'BEHAVIOR'. The main content area is titled 'Select anomaly result to change' and shows a table of 4 anomaly results. The table has columns: ID, BEHAVIOR LOG, RISK SCORE, CONFIDENCE, DECISION, and CREATED AT. The data rows are:

ID	BEHAVIOR LOG	RISK SCORE	CONFIDENCE	DECISION	CREATED AT
4	UserBehaviorLog object (4)	0.5	0.9	allow	July 10, 2025, 6:19 a.m.
3	UserBehaviorLog object (3)	0.5	0.9	allow	July 10, 2025, 6:12 a.m.
2	UserBehaviorLog object (2)	0.5	0.9	allow	July 10, 2025, 6:09 a.m.
1	UserBehaviorLog object (1)	0.5	0.9	allow	July 10, 2025, 6:06 a.m.

Below the table, it says '4 anomaly results'.

사용자 행동 로그에 대한 이상탐지 결과가 AnomalyResult 모델에 정상적으로 FK로 연결되어 관리되고 있음을 보여준다. risk_score, confidence, decision 필드가 저장됨을 확인하였다.

3) User Behavior Log 저장 및 관리 현황

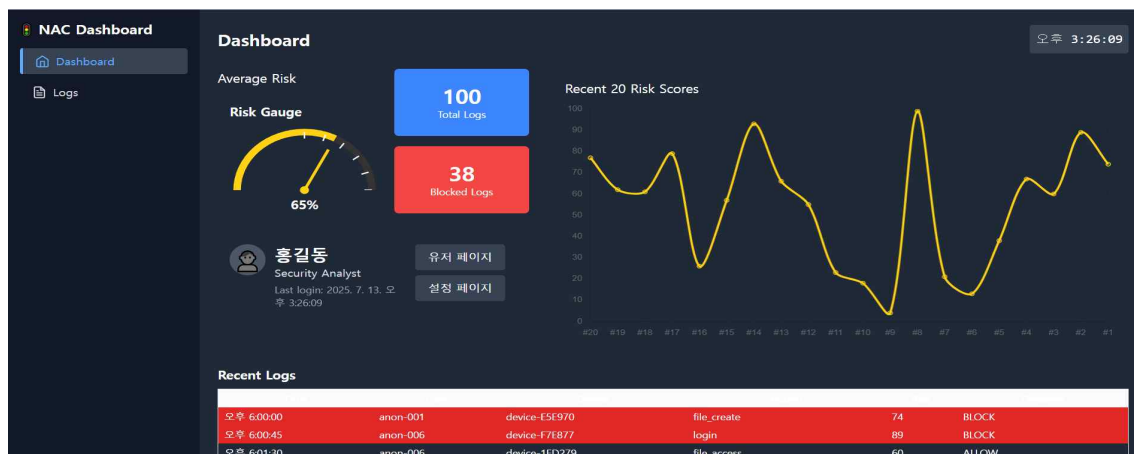


클라이언트에서 POST로 전송된 사용자 행동 로그가 UserBehaviorLog 모델에 기록되어 Admin 페이지에서 ID, user_id, action_type, request_time 등 주요 정보가 확인된다.

4.2 프론트엔드

백엔드에서 모델(LSTM-AE + Isolation Forest)이 실시간으로 판단한 이상 탐지 결과를 관리자가 한눈에 모니터링하고, 이상 분석 결과를 비교해 즉각 대응할 수 있도록 하는 NAC 대시보드이다.

1)NAC 대시보드 홈페이지



홈에 들어가면 이상 탐지 결과를 로그형식으로 볼 수 있고, 리스크 게이지를 자동차 속력게이지 형태로 시각화하여 직관성을 높였다. 전체 로그중 몇 개의 로그가 위험한 로그로 인식되었는지에 대한 컴포넌트도 배치되어있으며, 최근 20개의 로그의 위험도 점수를 그래프로 시각화 하여 관리자 친화적인 UI를 구성하고자 했다.

2)Logs 페이지

Time	User	Device	Action	Risk	Decision
오후 3:26:03	user_1	device_1	login	high	block
오후 3:26:03	user_2	device_2	download	low	allow
오후 3:26:03	user_3	device_3	login	low	allow
오후 3:26:03	user_4	device_1	download	high	allow
오후 3:26:03	user_5	device_2	login	low	block
오후 3:26:03	user_1	device_3	download	low	allow
오후 3:26:03	user_2	device_1	login	high	allow
오후 3:26:03	user_3	device_2	download	low	allow
오후 3:26:03	user_4	device_3	login	low	block
오후 3:26:03	user_5	device_1	download	high	allow
오후 3:26:03	user_1	device_2	login	low	allow
오후 3:26:03	user_2	device_3	download	low	allow
오후 3:26:03	user_3	device_1	login	high	block
오후 3:26:03	user_4	device_2	download	low	allow
오후 3:26:03	user_5	device_3	login	low	allow
오후 3:26:03	user_1	device_1	download	high	allow
오후 3:26:03	user_2	device_2	login	low	block
오후 3:26:03	user_3	device_3	download	low	allow
오후 3:26:03	user_4	device_1	login	high	allow
오후 3:26:03	user_5	device_2	download	low	allow

전체 로그와 위험로그로 판단된 로그를 볼 수 있는 페이지이다. 평범한 로그는 배경색과 맞추고, 위험 로그만 빨간색으로 더욱 돋보이게 UI를 구성했다. 더보기 버튼을 누르면 최근 로그를 20개씩 추가로 보여주게 된다.

3)아직 구현하지 못한 페이지 및 기능

3-1.유저 페이지

유저 페이지를 이용하여 유저의 정보를 수정할 수 있으며, 웹페이지 화면 UI도 사용자 설정으로 맞출 수 있는 기능을 제공할 예정이다.

3-2.관리자 페이지(정책 변경)

로그를 바탕으로 유동적인 정책 변경을 할 수있게 해주는 관리자용 페이지이다. 로그의 특성을 바탕으로 정책을 변경하여 이를 백엔드에 넘겨주는 과정을 거치게 된다. 관리자 페이지 접근에는 추가인증을 거치도록 조치할 예정이다.

3-3.시간대별 로그 조회기능

지금은 로그를 최근 20개씩 보여주고 있으나, 시간대별, 혹은 blocked 된 로그만 보고 싶은 순간이 생길 수도 있다. 이를 위해 디바이스 종류별, 시간대별 로그 조회를 가능하게 할 예정이다.

4.3 앱 어플리케이션

1)현재 수집중인 데이터 현황

센서 데이터:

1.가속도계

SensorManager.TYPE_ACCELEROMETER 활용

수집필드: x, y, z, magnitude, activity_level

수집 주기: 30초마다 스냅샷

데이터 형식: Float 배열 -> JSON 변환

2.자이로스코프

SensorManager.TYPE_GYROSCOPE 활용

수집필드: rotation_x, rotation_y, rotation_z, rotation_magnitude

수집 주기: 30초마다 스냅샷

데이터 형식: Float 배열 -> JSON 변환

3.근접센서

SensorManager.TYPE_PROXIMITY 활용

수집필드: distance, max_range, is_near, usage_context

수집 주기: 30초마다 스냅샷

데이터 형식: Float 배열 -> Boolean 변환 -> JSON 변환

시스템 상태 데이터:

1.배터리

BatteryManager 활용

```
val batteryFilter = IntentFilter(Intent.ACTION_BATTERY_CHANGED)
```

수집 데이터: level, scale, percentage, status, temperature, is_charging

2.네트워크

ConnectivityManager 활용

```
val networkCapabilities = connectivityManager.getNetworkCapabilities(activeNetwork)
```

수집 데이터:is_connected, has_wifi, has_cellular, has_ethernet

3.메모리

ActivityManager 활용

```
val memoryInfo = ActivityManager.MemoryInfo()
```

```
activityManager.getMemoryInfo(memoryInfo)
```

수집 데이터:available_memory, total_memory, memory_percentage,

low_memory

4.화면 밝기

Settings.System 활용

```
val brightness = Settings.System.getInt(contentResolver,  
Settings.System.SCREEN_BRIGHTNESS)
```

수집 데이터: brightness, brightness_percentage

5.앱 상태

자체 구현 메트릭

수집 데이터: session_duration, total_events, queue_size

사용자 인터랙션 데이터:

1.버튼 클릭

View.OnClickListener 활용

```
view.setOnTouchListener { v, event ->  
// 수집 데이터: x, y, pressure, size, duration, screen  
}
```

2.스크롤 이벤트

View.OnScrollChangeListener 활용

```
logText.setOnScrollChangeListener { _, scrollX, scrollY, oldScrollX,  
oldScrollY ->  
// 수집 데이터: scroll_direction, scroll_distance, scroll_speed,  
current_position  
}
```

3.버튼 클릭

View.OnClickListener 활용

```
button.setOnClickListener { view ->  
// 수집 데이터: button_type, screen_x, screen_y, view_width, view_height  
}
```

4.앱 생명 주기

Application.ActivityLifecycleCallbacks 활용

```
override fun onActivityStarted(activity: Activity) {  
// 수집 데이터: activity_name, task_id, duration  
}
```

전송 데이터 형식:

```
json
{
  "logs": [
    {
      "sequence_index": 45,
      "timestamp": "2025-07-12T09:30:00.000Z",
      "user_id": "device--1020935285-anonymous",
      "session_id": "session-uuid-abc123",
      "action_type": "snapshot_accelerometer",
      "params": {
        "x": -0.123,
        "y": 9.81,
        "z": 0.456,
        "magnitude": 9.813456,
        "activity_level": "gentle_movement",
        "snapshot_time": 1720778400000
      },
      "device_info": {
        "model": "Google sdk_gphone16k_x86_64",
        "os_version": "Android 16",
        "app_version": "1.0",
        "screen_density": 2.75,
        "network_type": "WiFi"
      },
      "location": {
        "latitude": 37.4219999,
        "longitude": -122.0840575,
        "accuracy": 20.0
      }
    },
    {
      "batch_size": 9,
      "timestamp": "2025-07-12T09:30:00Z",
      "source": "android-app"
    }
  ]
}
```

테스트 서버 응답 처리

json

```
{
  "status": "success",
  "decision": "allow",
  "risk_score": 0.15,
  "confidence": 0.92,
  "message": "정상 패턴입니다",
  "next_check_interval": 30,
  "processed_logs": 9,
  "total_received": 156,
  "server_timestamp": "2025-07-12 17:30:57"
}
```

2) 현재 UI/UX

1. 홈 화면



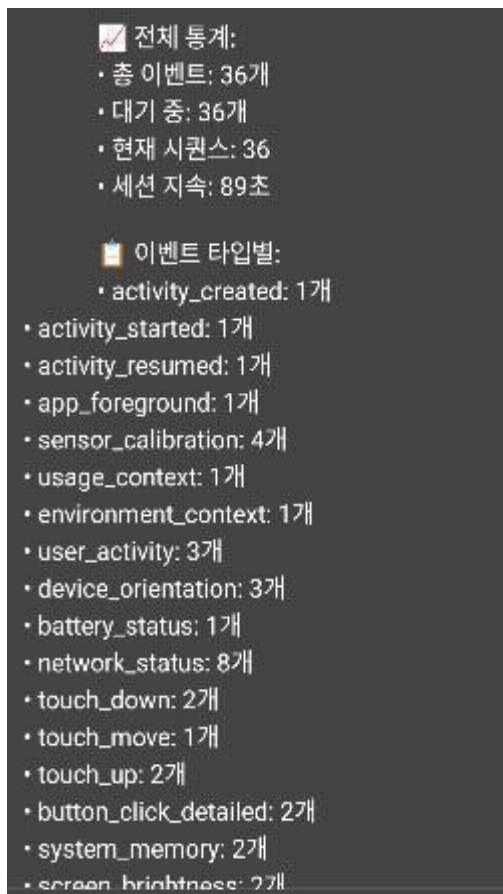
즉시 전송 버튼: 현재까지 쌓인 데이터를 서버로 전송

설정 버튼



데이터 초기화: 현재까지 쌓인 데이터를 초기화

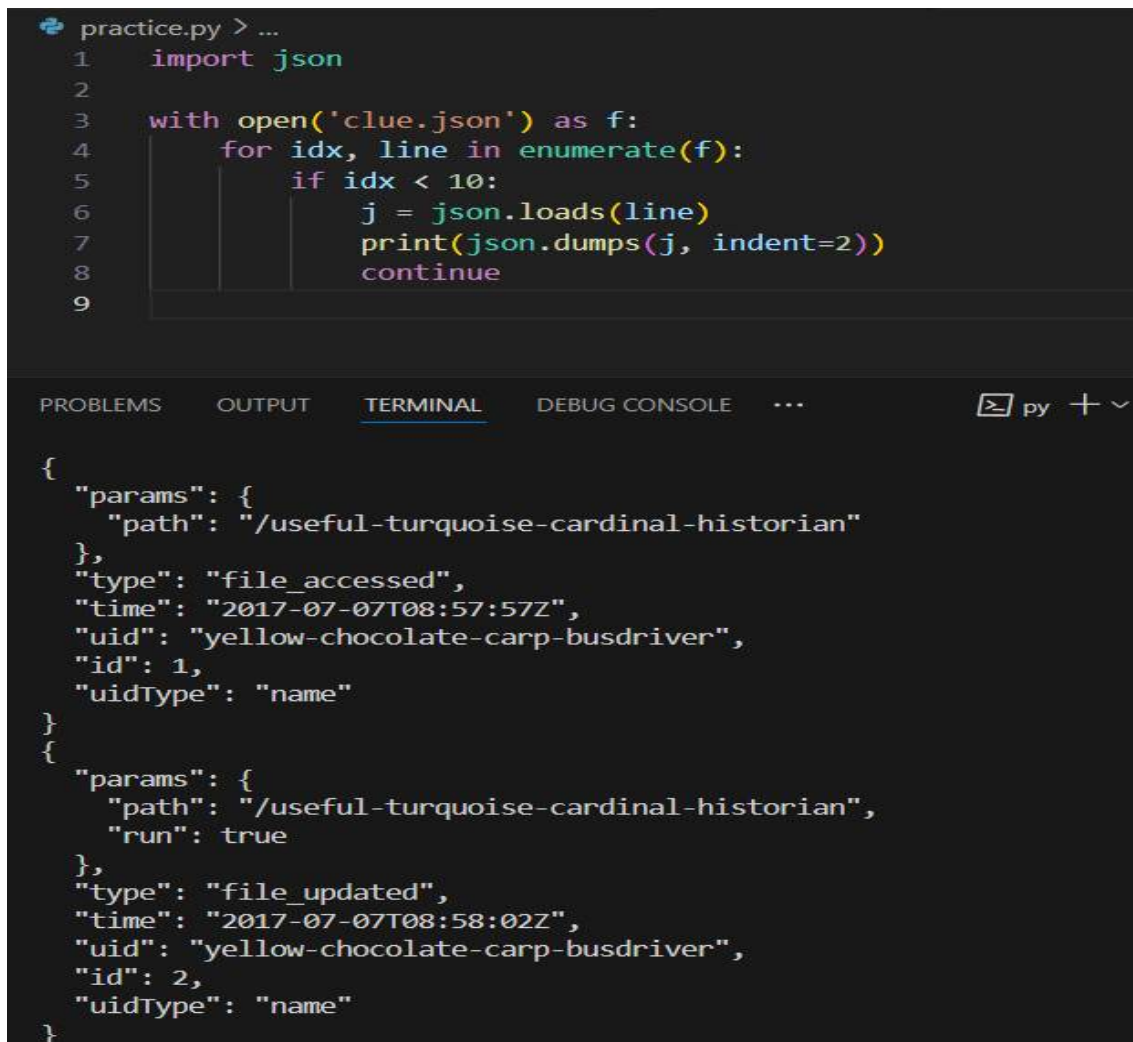
수집 통계 보기: 총 이벤트 및 데이터 타입별 개수 표기(아래 그림)



4.4 머신러닝 모델

0) CLUE-LDS 데이터셋 전처리 및 로그 분석

CLUE-LDS 데이터 셋의 로그데이터에는 여러 파라미터들이 존재 한다. 이러한 파라미터를 그대로 사용할 수는 없기 때문에 데이터 전처리 과정을 거쳐야 한다. 아래는 CLUE-LDS의 실제 로그 데이터 예시이다.



```

practice.py > ...
1  import json
2
3  with open('clue.json') as f:
4      for idx, line in enumerate(f):
5          if idx < 10:
6              j = json.loads(line)
7              print(json.dumps(j, indent=2))
8              continue
9

```

```

{
  "params": {
    "path": "/useful-turquoise-cardinal-historian"
  },
  "type": "file_accessed",
  "time": "2017-07-07T08:57:57Z",
  "uid": "yellow-chocolate-carp-busdriver",
  "id": 1,
  "uidType": "name"
}
{
  "params": {
    "path": "/useful-turquoise-cardinal-historian",
    "run": true
  },
  "type": "file_updated",
  "time": "2017-07-07T08:58:02Z",
  "uid": "yellow-chocolate-carp-busdriver",
  "id": 2,
  "uidType": "name"
}

```

이러한 파라미터를 전처리 하는 과정이 필요하기에 vectorize_clue.py로 전처리를 진행해 주었다. 데이터를 벡터화 하여 좀 더 빠른 데이터 처리를 위함이다.

```

vectorize_clue.py > ...
1
2 import json
3 import numpy as np
4 from datetime import datetime
5 import dateutil.parser
6 seq_len = 10
7 slide_size = 5
8
9 seq_data = []
10 uid_buffer = {}
11
12 with open('clue.json') as f:
13     for idx, line in enumerate(f):
14         j = json.loads(line)
15         uid = j['uid']
16         dt = dateutil.parser.isoparse(j['time'])
17
18         vector = [
19             1 if j['type'] == 'file_accessed' else 0,
20             dt.hour / 24.0,
21             1 if dt.weekday() < 5 else 0,
22             1 if '/sensitive/' in j.get('params', {}).get('path', '') else 0,
23             float(j['id']) / 100000.0
24         ]
25         if uid not in uid_buffer:
26             uid_buffer[uid] = []
27         uid_buffer[uid].append(vector)
28         if idx % 10000 == 0:
29             print(f'Processed {idx} lines, current seq: {len(seq_data)}')
30
31         if len(uid_buffer[uid]) >= seq_len:
32             seq_data.append(uid_buffer[uid][:seq_len])
33             uid_buffer[uid] = uid_buffer[uid][slide_size:]
34
35 np.save('clue_sequences.npy', np.array(seq_data))
36 print(f'Total sequences: {len(seq_data)}')
37 print(f'Saved clue_sequences.npy with shape: {np.array(seq_data).shape}')

```

vectorize_clue.py

```

Processed 50440000 lines, current seq: 10083946
Processed 50450000 lines, current seq: 10085946
Processed 50460000 lines, current seq: 10087941
Processed 50470000 lines, current seq: 10089935
Processed 50480000 lines, current seq: 10091931
Processed 50490000 lines, current seq: 10093915
Processed 50500000 lines, current seq: 10095908
Processed 50510000 lines, current seq: 10097899
Processed 50520000 lines, current seq: 10099896
Total sequences: 10100477

```

이렇게 전처리 된 CLUE-LDS 데이터 셋을 가지고 모델개발을 진행해 보았다.

1) LSTM-AE 개발 현황

데이터 셋을 벡터화 한 뒤, LSTM-AE 모델을 학습하여 이로 만들어진 모델의 성능을 검증하는 과정이 필요하다. train_latm_ae.py를 이용하여 정상 시퀀스를 학습시켰고,

이를 lstm_ae_ckpt.pt에 세이브하였다.

```

1  import torch
2  import torch.nn as nn
3  from torch.utils.data import DataLoader, TensorDataset
4  import numpy as np
5  device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
6
7  class LSTMAutoEncoder(nn.Module):
8      def __init__(self, input_dim, hidden_dim):
9          super().__init__()
10         self.encoder = nn.LSTM(input_dim, hidden_dim, batch_first=True)
11         self.decoder = nn.LSTM(hidden_dim, input_dim, batch_first=True)
12
13     def forward(self, x):
14         _, (h, _) = self.encoder(x)
15         h_repeat = h.repeat(x.size(1), 1, 1).permute(1,0,2)
16         out, _ = self.decoder(h_repeat)
17         return out
18
19 x = np.load('clue_sequences_scaled.npy')
20
21 x = torch.tensor(x, dtype=torch.float32)
22
23 dataset = DataLoader(
24     TensorDataset(X),
25     batch_size=256,
26     shuffle=True,
27     num_workers=2
28 )
29 model = LSTMAutoEncoder(5, 16).to(device)
30 opt = torch.optim.Adam(model.parameters(), lr=0.001)
31 criterion = nn.MSELoss()
32
33 total_batches = len(dataset)

```

train_lstm_ae.py의 일부 코드

이렇게 학습된 모델의 성능을 시험해보기 위해 make_test_sequences를 이용하여 정상+이상데이터가 섞인 테스트 시퀀스를 생성했다.

```

1  import json
2  import numpy as np
3
4  seq_len = 10
5  vectors = []
6  anomaly_flags = []
7
8  with open('clue_anomaly.json') as f:
9      for line in f:
10         j = json.loads(line)
11
12         vec = [
13             j.get('login_success', 0),
14             j.get('is_local_ip', 0),
15             j.get('download_size', 0) / 1e6,
16             j.get('domain_score', 0),
17             j.get('hour', 0) / 24.0
18         ]
19         vectors.append(vec)
20
21         is_anomaly = 0
22         if j.get('domain_score', 0) >= 0.45: is_anomaly = 1
23         if j.get('download_size', 0) >= 15e6: is_anomaly = 1
24
25         anomaly_flags.append(is_anomaly)
26
27 print(f"anomaly_flags: 이상치 총 {sum(anomaly_flags)}개")
28 sequences = []
29 labels = []
30 for i in range(len(vectors) - seq_len + 1):
31     seq = vectors[i:i+seq_len]
32     label = 1 if max(anomaly_flags[i:i+seq_len]) > 0 else 0
33     sequences.append(seq)
34     labels.append(label)
35 np.save('test_sequences.npy', sequences)
36 np.savetxt('labels.txt', labels, fmt='%d')
37 print(f"시퀀스 저장: {len(sequences)}개, 이상 시퀀스 {sum(labels)}개")

```

make_test_sequences.py

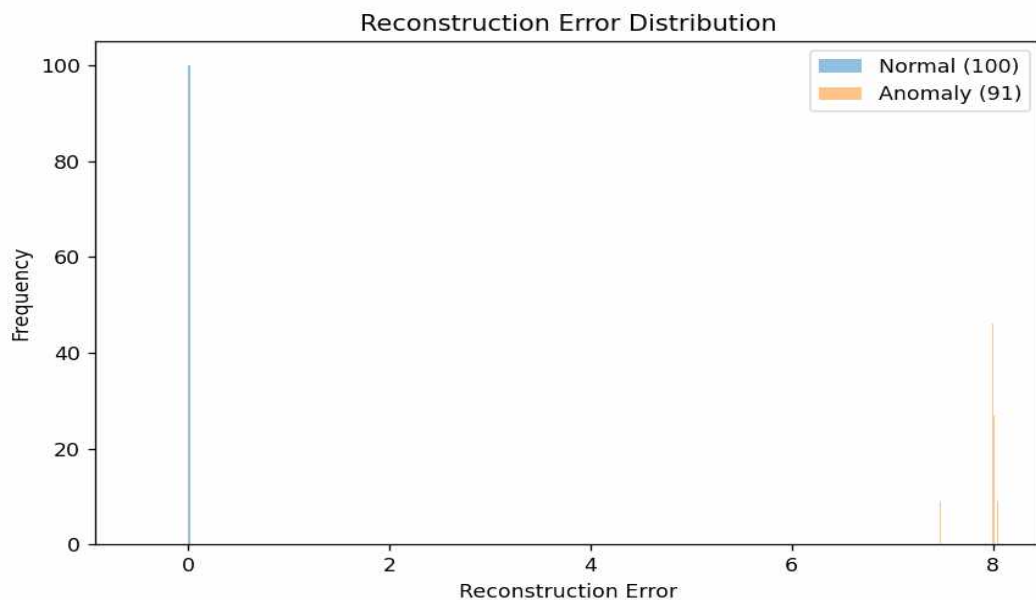
이후 재구성 오차를 구성, threshold를 조정하는 과정을 거친 뒤 f-1 score 등의 평가 지표를 이용하여 모델의 성능을 평가해 보았다.

```

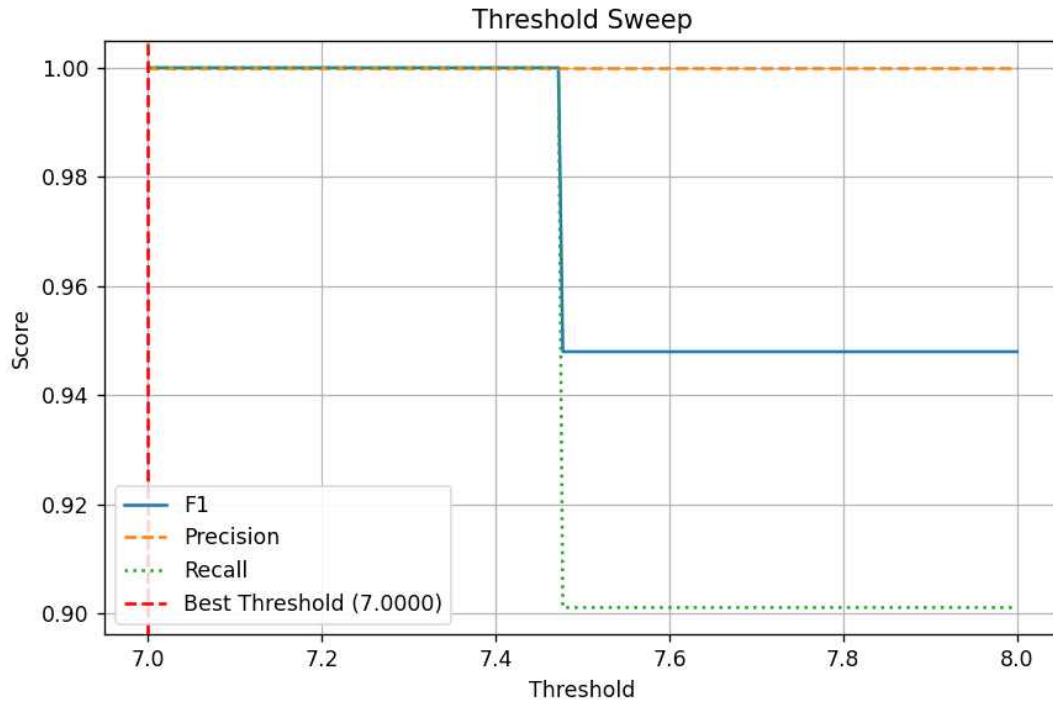
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from sklearn.metrics import precision_score, recall_score, f1_score
4
5  errors = np.loadtxt('recon_errors.txt')
6  labels = np.loadtxt('labels.txt')
7
8  normal = errors[labels == 0]
9  anomaly = errors[labels == 1]
10 mean_normal = np.mean(normal)
11 std_normal = np.std(normal)
12
13 max_candidate = mean_normal + 3 * std_normal
14 sweep_min = 7.0
15 sweep_max = 8.0
16
17 plt.figure(figsize=(8,5))
18 plt.hist(normal, bins=50, alpha=0.5, label=f'Normal ({len(normal)})')
19 plt.hist(anomaly, bins=50, alpha=0.5, label=f'Anomaly ({len(anomaly)})')
20 plt.title('Reconstruction Error Distribution')
21 plt.xlabel('Reconstruction Error')
22 plt.ylabel('Frequency')
23 plt.legend()
24 plt.show()
25
26 print(f"오차 범위: {errors.min():.6f} ~ {errors.max():.6f}")

```

run.py의 일부 코드



정상 데이터와 이상 데이터를 시각화



f-1 score, precision, recall등을 이용한 모델 평가

2) Isolation Forest 개발 현황

Isolation Forest의 성능평가를 위해서는 단일 테스트 시퀀스를 만들어야 할 필요가 있었다. 그래서 단일 테스트 시퀀스 json 파일을 만들었다.

아래는 단일 시퀀스 입력에 맞춰진 Isolation Forest 모델의 프로토타입이다.

```

41
42 iso_forest = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
43 iso_forest.fit(X_train)
44
45 joblib.dump(iso_forest, MODEL_PATH)
46 print(f'[INFO] Model saved to {MODEL_PATH}')
47
48
49 with open(SINGLE_LOG_PATH, encoding='utf-8') as f:
50     line = f.readline().strip()
51     j = json.loads(line)
52     dt = dateutil.parser.isoparse(j['time'])
53     test_vector = [
54         1 if j['type'] == 'file_accessed' else 0,
55         dt.hour / 24.0,
56         1 if dt.weekday() < 5 else 0,
57         1 if '/sensitive/' in j.get('params', {}).get('path', '') else 0,
58         float(j['id']) / 100000.0
59     ]
60
61 test_vector = np.array(test_vector).reshape(1, -1)
62 pred = iso_forest.predict(test_vector)
63 score = iso_forest.decision_function(test_vector)
64
65 print("\n[INFO] 단일 로그 판단 결과")
66 print(f"Vector : {test_vector.flatten().tolist()}")
67 print(f"Predict: {pred[0]} (1=정상, -1=이상)")
68 print(f"Score : {score[0]:.6f}")

```

isolation_forest_clue.py의 일부 코드

저 모델에 단일 시퀀스를 넣으면 모델이 그 로그가 정상인지(1), 비정상인지(-1) 판단하게 된다. 아래는 모델에 비정상 로그를 삽입했을 때의 결과 화면이다.

```
[INFO] Model saved to isoforest_model.pkl  
  
[INFO] 단일 로그 판단 결과  
Vector : [1.0, 0.08333333333333333, 1.0, 1.0, 9.99999]  
Predict: -1 (1=정상, -1=이상)  
Score  : -0.109789
```

5. 구성원별 진척도

팀원	역할
권태현	<ul style="list-style-type: none"> -머신러닝 모델(LSTM-AE 및 Isolation Forest)프로토타입 개발 완료 -머신러닝 모델의 성능 향상을 위한 리팩토링 진행중 -CLUE-LDS 데이터 전처리 완료 -사용자 로그 데이터 분석 진행중 -이상 탐지 결과 시각화 및 사용자 편의를 위한 대시보드(프론트엔드)프로토타입 개발 완료 -대시보드 기능 추가 진행중
구현서	<ul style="list-style-type: none"> -모바일 어플리케이션 UI 개발 완료 -어플리케이션 기능 추가 진행중 -사용자 행동 데이터 수집 진행중 -머신러닝 모델 경량화 방안 구상중
이승원	<ul style="list-style-type: none"> -사용자 행동 데이터 수신 및 처리를 위한 DB 프로토타입 개발 완료 -정책 관리 DB 프로토타입 개발완료 -Django를 이용한 백엔드 아키텍처 설계 및 보완중 -이상 탐지 모델 및 사용자 어플리케이션과의 연동 방안 구상중

6. 향후 추진 계획

