TECHNICAL REPORT · V3.0

Mobile Integrity Checker 프로젝트 보고서

모바일 앱 무결성 검증 시스템 — 루팅·탈옥·에뮬레이터·위변조 탐지 종합 솔루션

작성일	버전	플랫폼	분류
2026. 04. 01.	v3.0.0	Android	기술 보고서

TOC 목차 (Table of Contents)

01 프로젝트 개요

🎯 배경 및 목적

모바일 금융·게임·기업 앱은 루팅된 기기, 에뮬레이터, 위변조된 앱에서 실행될 때 심각한 보안 위협에 노출됩니다.

Mobile Integrity Checker는 이러한 위협을 실시간으로 탐지하고 서버에 보고하는 종합 무결성 검증 SDK입니다.

- 금융 앱 보안 규제(전자금융감독규정) 준수
- 게임 핵·치트 방지
- 기업 MDM 정책 위반 기기 차단
- 앱 리패키징·크래킹 방지

👤 타겟 사용자 (Target Audience)

우리 솔루션은 모바일 앱의 신뢰성이 중요한 모든 서비스 제공자를 대상으로 합니다.

- 금융 및 핀테크 보안 팀: 앱 위변조를 통한 부정 결제 및 계정 탈취를 방지하고자 하는 기업.
- 모바일 게임 서비스 운영사: 루팅/에뮬레이터를 이용한 핵(Hack) 툴 사용 및 게임 내 재화 복사를 차단하려는 개발사.
- 기업용 보안 솔루션 관리자 (MDM): 사내 업무용 앱이 보안이 취약한(루팅된) 기기에서 실행되는 것을 모니터링하려는 기업.
- 콘텐츠 서비스 제공자: OTT 또는 웹툰 등 유료 콘텐츠의 화면 캡처 및 유출을 방지하려는 플랫폼.

📝 사용자 시나리오 (User Scenarios)

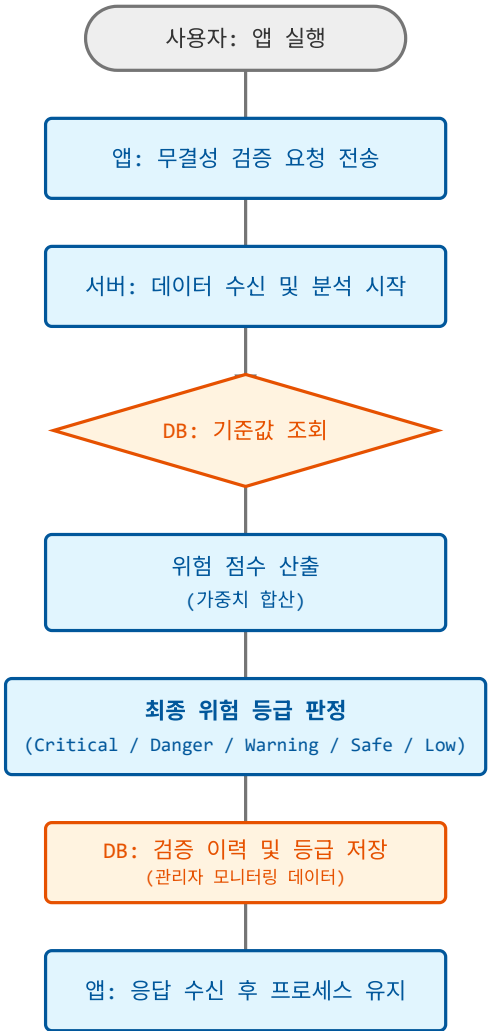
Scenario A: 악의적인 공격자에 의한 앱 위변조

- 1. 공격자: 금융 앱의 APK를 추출하여 결제 금액을 변조한 뒤 리패키징함.
- 2. 시스템: 앱 실행 시 서버가 apkHash와 signatureHash를 대조하여 위변조를 즉시 감지함.
- 3. 결과: 서버 응답으로 Critical 등급을 반환함.

Scenario B: 분석 도구를 이용한 로직 탈취 시도

- 1. 공격자: Frida를 실행하여 앱의 암호화 로직을 실시간으로 가로채려 시도함.
- 2. 시스템: 서버는 isFrida: true 신호를 수신하여 위험도를 100점으로 산출함.
- 3. 결과: 해당 기기를 Critical 상태로 판정하여 데이터 유출 사고를 사전에 방지함.

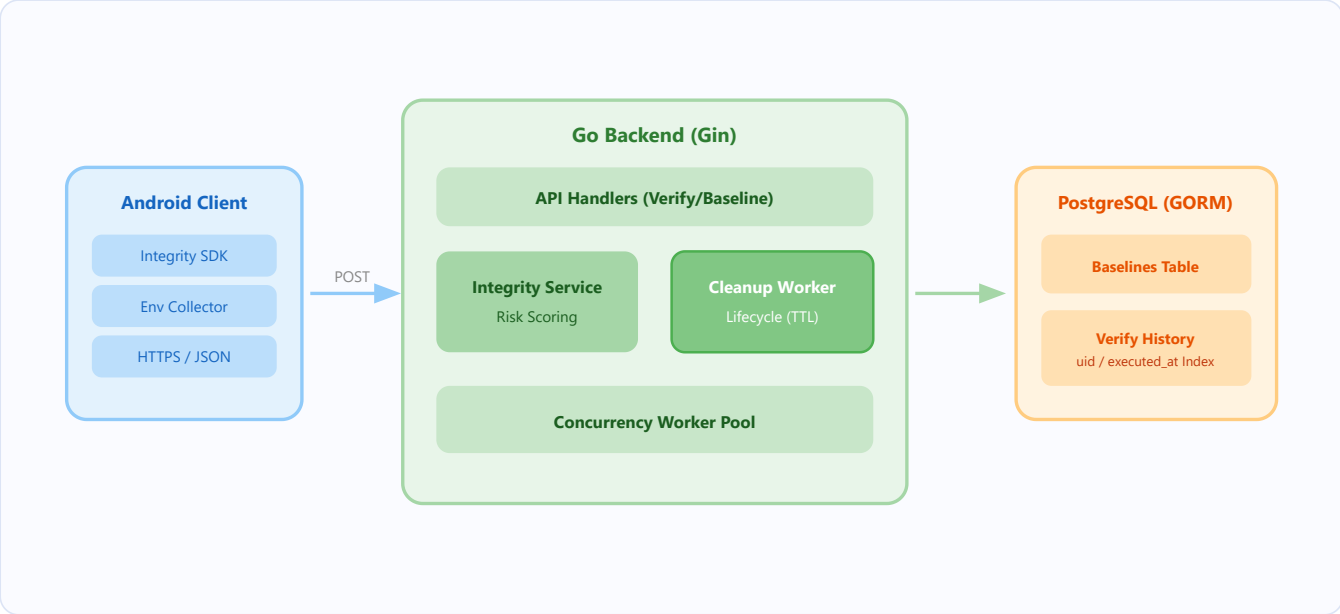
위험 등급 판정 프로세스



위험 등급 판정 및 데이터 관리 프로세스

02 시스템 아키텍처

전체 구조



기술 스택

클라이언트

Kotlin / Java | C++ NDK | HTTP/S

서버

Go(Golang) | Gin Framework | GORM | PostgreSQL

인프라

GitHub Actions | PostgreSQL Indexing

보안도구

gosec | govulncheck

03 핵심 기능

기능 ID	기능명	상세 내용
F-01	정적 무결성 검증	APK 해시 및 서명(Signature) 해시 값을 기준 데이터와 대조하여 앱의 물리적 변조 여부 확인.
F-02	동적 환경 분석	기기의 루팅(Rooting) 상태, 디버깅 모드 활성화 여부, 에뮬레이터 접속 여부 실시간 감지.
F-03	보안 도구 탐지	Frida, Xposed 등 메모리 변조 및 API 후킹에 악용되는 프레임워크의 실행 여부 확인.
F-04	위험도 기반 제어	탐지 항목별 가중치를 합산하여 위험 점수를 도출하고, 등급별 서비스 이용 허용(Allow) 여부 결정.

1. 정적 무결성 검증 (F-01)

- 설명: 앱의 바이너리(APK)와 서명값이 변조되었는지 확인합니다.
- 코드 연결: risk_service.go에서 ApkHash와 SignatureHash를 DB의 기준값과 대조하여, 불일치 시 **최상위 위험 등급인 Critical(100점)**으로 판정(Danger)하고, 검증 결과를 서버에 기록하여 부정 접근을 식별합니다.

2. 동적 환경 분석 (F-02)

- 설명: 루팅, 에뮬레이터, 디버깅 모드 등 앱이 실행되는 기기의 보안 상태를 실시간 감지합니다.
- 코드 연결: VerifyRequest 구조체로 전달된 Rooted, IsEmulator, DebugEnabled 등의 불리언(Boolean) 값을 체크하여 위험 점수를 합산하는 기능입니다 .

3. 보안 도구 탐지 (F-03)

- 설명: Frida, Xposed 같이 메모리 변조나 API 후킹에 사용되는 강력한 해킹 도구를 찾아냅니다.
- 코드 연결: risk_service.go에서 isFrida 탐지 시 Critical(100점), isXposed 탐지 시 Danger(80점) 등급으로 세분화하여 분류하는 핵심 방어선입니다 .

4. 위험도 기반 제어 (F-04)

- 설명: 여러 탐지 항목의 가중치를 합산해 최종 위험 등급을 매기고 서비스 이용 여부를 결정합니다.
- 코드 연결: RiskScore를 합산한 뒤 합산 점수에 따라 Critical(100점 이상), Danger(80점대), Warning(20~79점) 등으로 등급을 세분화하여 판정하고, 이 결과와 Allow 여부를 응답으로 반환하는 전체 프로세스입니다 .

04 API 상세 명세(API Specification)

🔍 API Request Handler

기능	API	Method	경로	설명
APK 무결성 검증 요청	VerifyHandler	POST	/api/verify	• 클라이언트가 전송한 APK 정보(패키지명, 해시값, 보안 환경 정보 등)를 JSON으로 수신(request.go) • 필수 필드 유효성 검사 수행 • 유효성 통과 시 Service 계층(VerifyApp)을 호출하여 위험도 분석 수행 • 분석 결과를 JSON으로 응답(response.go)
앱 기준값 등록	RegisterBaselineHandler	POST	/api/baseline	• 관리자가 전송한 앱의 정상 기준 정보(패키지명, APK 해시, 서명 해시)를 수신(base.go) • 필수 필드(packageName, apkHash, signatureHash) 유효성 검사 수행 • Repository 계층을 통해 Baseline 테이블에 저장
앱 기준값 조회	GetBaselineHandler	GET	/api/baseline/:packageName	• URL 경로 파라미터로 패키지명을 수신 • Repository 계층을 통해 해당 앱의 기준값 조회 • 미등록 앱인 경우 404 응답 반환
검증 이력 조회	GetResultsHandler	GET	/api/results	• 쿼리 파라미터로 UUID와 조회 기간(hours)을 수신 • Repository 계층을 통해 해당 기간 내 검증 결과를 조회하여 반환
위험 등급별 데이터 정리	CleanupHandler	POST	/api/cleanup	• Repository 계층의 등급별 데이터 정리 함수를 호출 • Safe/Low(7일), Warning/Danger(1개월), Critical(3개월) 경과 데이터를 자동 삭제

📦 api routing

```
api.POST("/verify", handler.VerifyHandler)
api.POST("/baseline", handler.RegisterBaselineHandler)
api.GET("/baseline/:packageName", handler.GetBaselineHandler)
api.GET("/results", handler.GetResultsHandler)
api.POST("/cleanup", handler.CleanupHandler)
```

🔍 Risk Analysis Engine & Scoring Logic

단계	핵심 컴포넌트	설명
요청 대기	VerifyQueue	유저로부터 접수된 앱 검증 요청 데이터를 Queue에 안전하게 보관하며 줄을 세움.
병렬 처리	Worker / StartWorkerPool	여러 개의 Worker(작업자)가 Queue에서 요청을 가져와 동시에(병렬적으로) 처리하여 서버의 과부하 방지.
위험도 평가	VerifyApp	루팅, 해킹 툴, 위변조 여부 등을 꼼꼼히 분석하여 해당 앱의 보안 위험 점수와 등급 판정
결과 저장	VerifyApp	모든 평가가 끝난 최종 결과 데이터는 DB(repository.SaveVerifyResult)에 기록하여 검증 이력을 영구적으로 보관

📊 탐지 항목 요약

카테고리	탐지 항목	위험도	산출 점수	비고(상세 로직)
앱 무결성	APK 해시 위변조	CRITICAL	100점	apkHash 불일치 감지
앱 무결성	서명(Signature) 위변조	CRITICAL	100점	signatureHash 불일치 감지
동적 분석	Frida 탐지	CRITICAL	100점	isFrida == "true"
동적 분석	Xposed 탐지	DANGER	80점	isXposed == "true"
기타	미등록 패키지	WARNING	50점	UnregisteredPackage == "true"
루팅	Rooting 기기 탐지	WARNING	40점	rooted == "true"
동적 분석	디버깅 모드 활성화	WARNING	20점	DebugEnabled == "true"
가상 환경	에뮬레이터 실행	LOW	10점	isEmulator == "true"

05 데이터베이스 설계 (Data Architecture)

ER-Diagram 및 테이블 설계

단순한 저장이 아닌 보안 데이터의 특성에 맞춘 효율적인 스키마를 설계

- Baselines (앱 기준 정보): 패키지명을 PK로 하여 고유성을 보장하고, apk_hash와 signature_hash를 저장하여 검증의 정답지로 활용.
- VerifyHistory (검증 이력): 사용자 기기 식별값(uid), 위험 점수(risk_score), 판정 등급(risk_level) 등을 기록하여 보안 감사 추적성 확보.

데이터 조회 성능 최적화 (Indexing)

"대량의 로그 데이터에서 특정 기기(uid)의 최근 상태(executed_at)를 O(log N) 시간 복잡도로 빠르게 추출하기 위해 전략적 인덱싱을 적용

- Composite Index 적용: uid와 executed_at 컬럼에 복합 인덱스를 생성.
- 기대 효과: 수만 건의 로그 데이터 중 특정 기간(hours 파라미터 기반)의 데이터 필터링 속도 개선.

[핵심] 데이터 생명주기 제어 (Log Lifecycle Management)

보안 등급에 따라 로그 보관 기간을 차등화하여 스토리지 효율 극대화

- 등급별 차등 보관 (TTL Policy):
  - Safe, Low: 7일 후 자동 삭제 (불필요한 정상 로그 최소화)
  - Warning: 30일 보관 (관찰 필요 데이터)
  - Danger, Critical: 90일 보관 (보안 사고 대응 및 분석용)
- 자동화 구현: CleanupByRiskLevel 함수를 통해 DB 레벨에서 DELETE 쿼리를 최적화하여 실행.

GORM 기반 인프라 자동화

- AutoMigrate: Go 코드의 구조체(Struct) 변경 사항을 DB 스키마에 즉시 반영하여 새로운 보안 탐지 항목(예: 새로운 해킹 툴) 추가 시 스키마 변경 비용을 최소화함.
- Transaction Safety:
  - 검증 결과 저장과 로그 정리가 안전하게 이루어지도록 GORM 인터페이스 활용.
  - 결과 저장과 동시에 이전 로그를 정리할 때 데이터 정합성을 보장하기 위해 GORM 트랜잭션을 활용.

등급별 데이터 생명주기

위험 등급	보관 기간	처리 목적
Safe, Low	7일	일반적인 정상 접근 로그 관리
Warning, Danger	1개월	반복적인 이상 징후 추적 및 분석
Critical	3개월	위험 추적 및 분석

06 위험도 산출 및 운영 정책 (Policy & Operations)

위험도 점수(RiskScore) 산출

각 탐지 항목별 위험 가중치를 부여한 후 합산 로직 적용.

```
// 2. 보안 환경 검사 (String -> Bool 변환 및 점수 합산)
isRooted := req.Rooted
isDebug := req.DebugEnabled
isFrida := req.IsFrida
isXposed := req.IsXposed
isEmulator := req.IsEmulator

if isRooted {
    riskScore += 40
}
if isFrida {
    riskScore += 100
}
if isXposed {
    riskScore += 80      Lesaowja, 4일 전 • 소스 연결
}
if isDebug {
    riskScore += 20
}
if isEmulator {
    riskScore += 10
}
```

위험 등급(RiskLevel) 체계

산출된 점수에 따라 Safe / Low / Warning / Danger / Critical 등급 판정.

등급	점수	보안 위협 수준 (Threat Level)	상세 판정 사유 (Technical Reasoning)
Safe	0점	🛡️ 무결성 유지	OS 샌드박스 보호 정상 작동. 메모리 및 파일 시스템 접근 통제 유지.
Low	1 ~ 19점	🔍 잠재적 위험	에뮬레이터 환경 실행 등 직접적인 공격 징후는 낮으나 정책상 모니터링이 필요한 비정상 실행 환경.
warning	20 ~ 79점	⚠️ 보안 경계 무력화	최고 관리자 권한 노출(루팅) 또는 분석 채널 개방(디버깅)으로 인해 언제든 메모리 변조 및 데이터 탈취가 가능한 취약 환경.
Danger	80 ~ 99점	🔴 시스템 변조	Xposed 프레임워크 등 시스템 레벨의 변조 도구가 활성화되어 앱 보안 기능 무력화가 감지된 상태.
Critical	100점 이상	🚨 무결성 파괴	APK/서명 위변조가 감지되거나 Frida를 통한 실시간 코드 주입(Hooking)이 확인된 명백한 침해 상황.

07 공급망 보안 및 품질 관리 (DevSecOps)

SBOM (Software Bill of Materials) 관리

- 목적: 사용 중인 오픈소스 라이브러리(Gin, GORM, Postgres Driver 등)의 투명성 확보 및 취약점 관리.
- 도구:govulncheck 를 활용하여 프로젝트의 의존성 패키지를 스캔해 공개 취약점(CVE) 탐지.

CI/CD 파이프라인 구성

- CI (gosec, GitHub Actions): 코드 Push 시 자동 빌드(예정), Unit Test, gosec 도구를 통한 취약점 자동 스캔.
- CD: 테스트 통과 시 운영 서버 자동 배포 프로세스.

보안 취약점 점검 내역

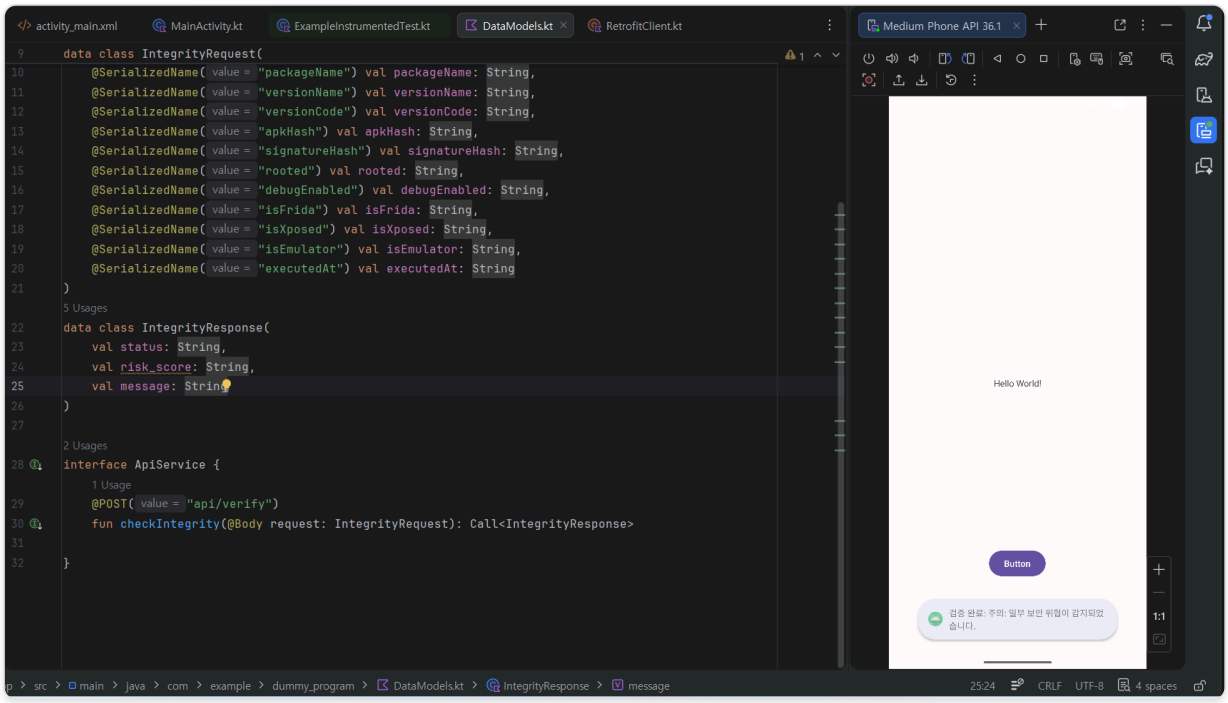
- govulncheck 분석 결과 기록 및 패키지 보안 업데이트 이력 상시 관리.
- GitHub Actions 로그 관리.

08 현재 진행 상황 및 향후 계획

현재 진행 상황

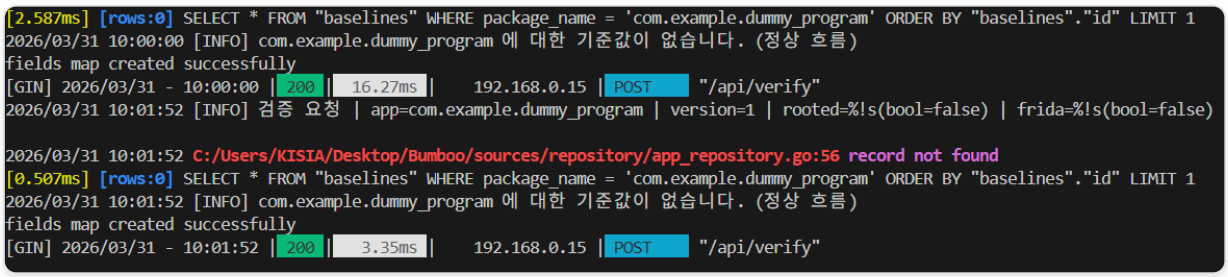
- 백엔드 보안 엔진 구축: Go(Gin) 기반의 5종 핵심 API 구현 및 무결성 검증 로직 검증 완료.
- DB 최적화 및 자동화: PostgreSQL 복합 인덱싱(uid, executed_at) 및 위험 등급별 로그 생명주기(TTL) 관리 로직 구현 완료.
- 로컬 보안 점검 수행: 개발 단계에서 gosec 및 govulncheck를 활용하여 소스코드 및 외부 라이브러리의 중대 취약점(Critical) 수동 점검 완료.

① 안드로이드 클라이언트 SDK 연동 및 API 통신 테스트



- Android Native 환경에서의 데이터 모델 구성 및 실시간 탐지 결과 확인

② Go 백엔드 서버 실시간 요청 처리 및 비즈니스 로직 로그



- Gin 프레임워크를 통한 엔드포인트 핸들링 및 데이터 무결성 검증 프로세스 실행

③ PostgreSQL 데이터베이스 검증 이력 적재 및 관리 현황

	id [PK] inte	uuid text	package_na text	version_ bigint	apk_has text	signature_hash text	rooted boolean	debug_ boolean	is_frida boolean	is_xposed boolean	is_emul boolean	risk_scc bigint	risk_level text	allow boolean	executed_at timestamp with time zone
1	10	c85ac3...	com.exa...	1	1d3f...	5dba8cc73b7d5fa4d...	false	true	false	false	true	80	Danger	false	2026-03-31 10:01:47+09
2	9	c85ac3...	com.exa...	1	1d3f...	5dba8cc73b7d5fa4d...	false	true	false	false	true	80	Danger	false	2026-03-31 09:59:56+09
3	8	c85ac3...	com.exa...	1	1d3f...	5dba8cc73b7d5fa4d...	false	true	false	false	true	80	Danger	false	2026-03-30 22:23:14+09

- 판정 결과(Risk Level), 허용 여부(Allow), 실행 시간 등 핵심 보안 지표 저장 확인

향후 추진 계획

- CI/CD 파이프라인 자동화 (GitHub Actions):
  - 현재 **수동**으로 수행 중인 보안 스캔(gosec, govulncheck)을 GitHub Actions 워크플로우로 이관하여 코드 Push 시 자동 점검 체계 구축 예정.
- 플랫폼 및 탐지 영역 확장:
  - iOS 환경 지원: Swift 기반의 iOS 전용 검증 모듈 및 시뮬레이터 탐지 로직 추가 개발.
  - 크로스 플랫폼 대응: Flutter, React Native 등 하이브리드 앱 환경을 위한 범용 보안 SDK 설계 및 배포.
- 실시간 성능 및 가시성 강화:
  - Redis 캐시 도입을 통한 기준값(Baseline) 조회 성능 극대화 및 보안 로그 시각화 대시보드 구축.

09 기대 효과 및 향후 과제

기술적 기대 효과

- 자동화된 로그 관리 시스템 구축 및 실시간 보안 위협 대응력 강화.
- DB 인덱스 최적화를 통해 실시간 검증 요청에 대한 응답 지연(Latency) 최소화.

확장 방향

- 실시간 보안 대시보드
- ios 기반 어플리케이션 개발
- API기반이 아닌 에이전트 기반 Apk개발
- 커널 레벨 탐지