홍진수 (경력 기술서)

https://kaameo.github.io

Github: https://github.com/kaameo
Email : kaameo12@gmail.com
Mobile : (+82) 010-9279-1819

1. DeFender X SIEM

- **소속**: (주)인브릿지 *2025. 06 - PRESENT*

주요 내용:

여러 벤더의 보안 침해 로그를 정규화하고 상관 관계 분석, 탐지 · 대응 및 보고서 생성을 자동화하는 프로젝트. 풀스택 개발 및 데이터 파이프라인 설계 · 구축을 담당. 6명 개발팀에서 진행.

사용한 기술: TypeScript, Next.js, Prisma, OpenSearch, PostgreSQL, Kafka, Docker, Kubernetes

주요 성과:

Palo Alto Cortex XDR, Cisco SecureX 등 여러 벤더의 보안 로그 수집 및 정규화 파이프라인 구축 / GitHub Actions 기반 CI/CD 파이프라인 구축으로 배포 리드타임 50% 단축 / 실시간 보안 위협 모니터링 대시보드 개발 / BlockNote 기반 리치 텍스트 에디터 및 커스텀 차트 블록, AI 에디터 기능 구현

- **1.1 데이터 파이프라인 구축**
  여러 보안 벤더들의 침해 로그를 수집하고 정규화하는 데이터 파이프라인을 구축하는 것을 목표로 둠. 구현한 기능들은 아래와 같음.

  #### 1.1.1. 데이터 수집
  Palo Alto XDR REST API 분석 결과, 다른 벤더와 다르게 실시간 스트리밍 방식 없이 페이징 방식만 제공하는 것을 확인함. 이를 해결하기 위해 로그의 modification_time 기반 폴링 수집기를 설계함. Redis에 마지막 수집 시점을 저장하고 주기적으로 변경분만 조회하는 증분 수집 패턴을 적용함.
  사용한 기술: Go, Redis

  #### 1.1.2. 데이터 전송
  Apache Kafka를 도입하여 수집기와 처리기를 분리함. 벤더별/테넌트별 토픽을 분리하고 파티션 키를 사건 ID로 지정하여 순서 보장과 병렬 처리를 동시에 달성함. Kafka 도입으로 인해 관리해야할 리소스가 늘었으나, 리소스 격리, 장애 격리, 확장성 측면에서 개선이 있었음.
  사용한 기술: Apache Kafka, Go

  #### 1.1.3. 처리 및 적재 (Processing & Storage)
  Kafka OpenSearch Sink Connector를 활용하여 이벤트 데이터를 OpenSearch에 적재함. OpenSearch의 인덱스/템플릿/ILM을 조정하여 저장 효율을 극대화함.
  사용한 기술: Kafka Connect, OpenSearch

  #### 1.1.4. Python에서 Go로 전환
  초기 Python으로 구현했으나, 대량 로그 처리 시 GIL(Global Interpreter Lock)로 인한 병렬 처리 한계와 높은 메모리 사용량 문제가 발생함. Go로 전환하여 goroutine 기반 동시성 처리로 처리량을 3배 향상시키고 메모리 사용량을 60% 절감함.
  사용한 기술: Go

  #### 1.1.5. 결과 및 학습
  엔드포인트별 CVE 데이터의 경우 매 수집 시 100MB임을 확인함. 매 시간 수집이 필요한지 요구사항을 재검토하여 하루 한 번 수집으로 변경하고 스토리지 사용량을 대폭 절감함. 도메인 특성(실시간성 요구 수준, 데이터 변화 빈도)을 정확히 파악하면 인프라와 데이터 구조의 근본적 최적화가 가능함을 확인함.
  또한 Kafka 도입으로 수집기와 처리기를 분리하여 확장성과 안정성을 확보함. 시스템 복잡도가 증가했으나, 장애 격리와 리소스 격리가 가능해져 운영 효율이 향상됨을 체감함.

- **1.2 배포 자동화**

  GitHub Actions 기반 CI/CD 파이프라인을 구축하여 배포를 자동화 하여 안정화 하고 배포 리드타임을 50% 단축하는 것을 목표로 둠.

  #### 1.2.1. 문제 분석

  로컬 개발 → 개발 브랜치 머지 → 서버 재가동의 수작업 루틴이 반복되어 빌드 에러/휴먼 에러에 따른 서버 가동 중단 위험이 있었음.

  사용한 기술: Git, GitHub

  #### 1.2.2. CI/CD 파이프라인 구성

  GitHub Actions로 머지/PR 트리거 시 자동 빌드 · 배포 파이프라인을 구성함. 인바운드가 차단되고 아웃바운드만 가능한 서버 환경을 고려해 self-hosted runner로 아웃바운드 중심의 액션 실행 구조를 설계함.

  사용한 기술: GitHub Actions, Docker, Kubernetes

  #### 1.2.3. 결과 및 학습

  배포 리드타임을 수동 10분에서 자동 5분으로 50% 단축함. 수동 재가동을 제거하여 배포 안정성을 향상시킴. 변경 이력과 리뷰가 체계화되어 회귀 이슈를 감소시키고 협업 품질을 개선함.

- **1.3 보안 대시보드 풀스택 개발**

  보안 운영팀이 실시간으로 위협 현황을 모니터링하고 신속하게 대응할 수 있는 보안 대시보드를 개발하는 것을 목표로 둠.

  #### 1.3.1. 요구사항 분석

  보안 운영팀의 워크플로우를 분석한 결과, 여러 벤더 콘솔을 개별 확인하는 방식으로 인해 위협 대응 시간이 지연되고 있었음. 통합 뷰와 실시간 알림, 빠른 필터링이 핵심 요구사항임을 도출함.

  사용한 기술: Next.js, TypeScript, Prisma

  #### 1.3.2. 프론트엔드 및 백엔드 구현

  빠른 MVP 개발을 위해 Next.js App Router 기반으로 대시보드 UI 와 백엔드를 구현함. OpenSearch에서 집계된 보안 이벤트를 조회하는 API를 설계하고 Prisma로 PostgreSQL 메타데이터를 관리함. 심각도/벤더/시간 범위별 필터링 및 페이지네이션을 적용함.

  사용한 기술: Next.js, TypeScript, Prisma, OpenSearch, PostgreSQL

  #### 1.3.3. 결과 및 학습

  벤더별로 분산되어 있던 보안 현황을 단일 대시보드로 통합하여 위협 식별 시간을 단축함. 프론트엔드와 백엔드를 함께 설계하면서 API 계약을 먼저 정의하는 것이 개발 효율과 유지보수성을 높이는 핵심임을 배움.

- **1.4 보고서 에디터 개발**

  보안 분석가가 보고서를 효율적으로 작성할 수 있는 리치 텍스트 에디터를 개발하는 것을 목표로 둠.

  #### 1.4.1. 요구사항 분석 및 기술 선정

  보고서 작성 UX 개선을 위해 BlockNote와 Markdown 에디터를 비교 분석함. 각 포맷으로 프로토타입을 구현하여 보안 분석가에게 피드백을 받은 결과, 블록 기반 편집이 직관적이라는 의견을 반영하여 BlockNote를 선정함.

  사용한 기술: Next.js, TypeScript, BlockNote

  #### 1.4.2. 커스텀 블록 및 AI 에디터 구현

  보고서에 차트 삽입이 필요하여 차트 커스텀 블록을 구현함. BlockNote 코어는 Apache 2.0 라이선스이나, @blocknote/xl-* 확장 패키지와 AI 에디터 기능은 GPL 3.0 라이선스로 상용 프로젝트에 적용이 어려웠음. 라이선스 문제를 해결하기 위해 차트 블록과 AI 에디터 기능을 직접 구현함. AI 에디터는 편집 중 AI의 의견을 받아 내용을 수정할 수 있도록 설계함.

  사용한 기술: Next.js, TypeScript, BlockNote, Recharts, OpenAI API

  #### 1.4.3. 결과 및 학습

라이선스 제약을 우회하여 상용 프로젝트에 필요한 기능을 확보함. 오픈소스 라이브러리 도입 시 코어와 확장 패키지의 라이선스를 분리하여 검토해야 함을 배움. 사용자 피드백 기반의 기술 선정이 채택률과 만족도를 높이는 핵심임을 확인함.

2. Teams Add-in 개발

- **소속**: (주)인브릿지 *2025.03 - 2025.05*

주요 내용:

Microsoft Teams 통화/미팅 자동 녹취 및 LLM 요약 기능을 제공하는 Add-in 프로젝트에 중도 합류. 프론트엔드 위주 풀스택 개발을 담당. 4명 개발팀에서 진행.

사용한 기술: ASP .NET, JavaScript, React, Node.js, Express, Supabase(PostgreSQL), Docker, Teams Add-in

주요 성과:

중도 합류 후 빠르게 온보딩하여 기능 개발 및 이슈 처리 / 로그성 테이블 스키마 단순화 및 쿼리 지연 감소 / GitHub 기반 협업 체계 구축으로 프로덕션 직접 수정 위험 제거 / Docker 기반 개발 환경 표준화로 신규 팀원 온보딩 시간 단축

- **2.1 협업 체계 구축 및 프로덕션 리스크 제거**
  GitHub 도입을 통해 브랜치 기반 협업 체계를 구축하고 프로덕션 직접 수정 위험을 제거하는 것을 목표로 둠.

  2.1.1. 문제 분석
  상용 서버에서 직접 소스를 수정하던 상황에서 인원이 증가하며 overwrite와 충돌이 빈번하게 발생함. 프로덕션 직접 수정으로 인한 롤백 불가 등의 리스크가 컸음.
  사용한 기술: Git

  2.1.2. 협업 체계 구축
  GitHub를 도입하여 브랜치 기반 협업 체계(기능 브랜치 → 코드 리뷰 → 머지)를 구축함. Teams Auth 플로우를 분석하고 Mock/개발 설정을 정리하여 로컬 실행 및 디버깅 환경을 마련함. Notion 도입을 주도하여 산출물 및 트러블슈팅 템플릿을 표준화함.
  사용한 기술: GitHub, Notion, Docker

  2.1.3. 결과 및 학습
  서버 직접 수정을 중단하여 충돌과 오류를 감소시키고 변경 이력 추적이 가능해짐. 신규 합류자의 온보딩 시간을 단축하고 재발 사고를 예방함. 협업 도구를 목적에 맞게 활용하는 것이 생산성과 제품 품질에 직결된다는 것을 체감함.

- **2.2 도메인 모델 재설계 및 API 표준화**
  도메인 모델을 재설계하고 API 스펙을 표준화하여 협업 비용을 절감하는 것을 목표로 둠.

  2.2.1. 문제 분석
  통화 녹취 기능 개발 중 Caller와 Callee를 개별 소유 구조로 관리하면서 권한 처리와 조인 로직의 복잡도가 증가하는 문제가 있었음. 또한 각자 목록 화면을 개발하면서 페이지네이션 방식이 불일치하는 문제가 발생함.
  사용한 기술: PostgreSQL, Supabase

  2.2.2. 스키마 재설계 및 API 표준화
  'user(사용자)/counterpart(상대방) + 수신/발신 플래그' 구조로 스키마를 재설계하는 RFC를 작성함. 변경 이유와 장단점을 문서화하여 팀 합의를 이끌어냄. API 스펙 표준(정렬, 사이즈, 페이지 토큰 규격)을 제안하고 팀 규약으로 통일함.
  사용한 기술: PostgreSQL, Node.js, Express, React

  2.2.3. 결과 및 학습
  단계적 마이그레이션을 통해 API와 권한 로직을 단순화함. 중복되던 컴포넌트를 공용화하여 협업 비용을 절감함. 도메인 정의를 먼저 맞추는 것이 사이드이펙트를 줄이고 협업 효율을 높이는 핵심임을 배움.

3. 개인 블로그

- **소속**: 사이드 프로젝트 *2025.07 - 현재*

  주요 내용:

  기술 학습과 경험 공유를 위한 Next.js 기반 기술 블로그 플랫폼. 1인 개발.

  사용한 기술: Next.js 14, TypeScript, Tailwind CSS, shadcn/ui, GitHub Actions, GitHub Pages

  주요 성과:

  Next.js SSG 기반 정적 개인 블로그 구축 / MDX 기반 콘텐츠 관리로 DB 없이 Git 버전 관리 실현 / GitHub Actions CI/CD 파이프라인 구축으로 자동 배포 환경 구성

  링크: https://kaameo.github.io/ | GitHub Repository

  - **3.1 정적 블로그 아키텍처 설계**
    텍스트 기반의 가장 간단한 아키텍처를 설계하는 것을 목표로 둠.

    3.1.1. 요구사항 분석 및 기술 선정
    블로그 콘텐츠의 특성(변경 빈도 낮음, SEO 중요, 빠른 로딩 필요)을 분석하여 SSR 대신 SSG 방식을 선택함. Next.js의 정적 export 기능과 GitHub Pages 무료 호스팅을 조합하여 운영 비용 없는 구조를 설계함.
    사용한 기술: Next.js 14, TypeScript

    3.1.2. MDX 기반 콘텐츠 관리 시스템 구축
    별도 CMS나 데이터베이스 없이 프로젝트 내 MDX 파일로 포스트를 관리하는 구조를 구현함. Git 기반 버전 관리로 콘텐츠 변경 이력 추적이 가능하고, 마크다운 내 React 컴포넌트 삽입으로 인터랙티브한 콘텐츠 작성이 가능함.
    사용한 기술: MDX, gray-matter, next-mdx-remote

    3.1.3. 결과 및 학습
    DB 없이 Git만으로 콘텐츠를 관리하여 인프라 복잡도와 운영 비용을 제거함. 텍스트 위주의 콘텐츠 특성에 맞는 렌더링 전략 선택이 성능과 비용 모두에 영향을 미친다는 것을 체감함.

  - **3.2 CI/CD 파이프라인 및 배포 자동화**
    GitHub Actions를 활용하여 배포 자동화 환경을 구축하는 것을 목표로 둠.

    3.2.1. 배포 파이프라인 구성
    main 브랜치 푸시 시 자동으로 빌드 및 배포가 실행되는 GitHub Actions 워크플로우를 구성함. Next.js 정적 export 후 GitHub Pages로 배포되도록 설정함.
    사용한 기술: GitHub Actions, GitHub Pages

    3.2.2. 결과 및 학습
    글 작성 후 커밋만으로 자동 배포되어 운영 부담을 최소화함. 정적 사이트의 경우 GitHub Pages와 Actions 조합으로 무료로 완전한 CI/CD 환경을 구축할 수 있음을 확인함.