김현욱 Hyunwook Kim

서로 다르게 표기된 데이터를 같은 것으로 식별·연결해온 Data Engineer

Phone: 010-5580-2468 Email: hyunwook711@naver.com
GitHub: github.com/hyunwook711 Portfolio: portfolio.hwlabs.dev

소개

저는 서로 다르게 표기된 데이터를 같은 것으로 식별하고 연결하는 일을 해온 데이터 엔지니어입니다. 소프트웨어 공급망 보안 제품에서 Maven·npm·PyPI 등 생태계마다 다른 패키지 표기를 하나의 컴포넌트로 매핑하고, CVE·OSV와 12종 Linux 배포판의 보안 권고안을 제품·버전 범위에 연결했습니다. 취약점 데이터는 수집됐다는 이유만으로 정답이 되지 않기 때문에, 릴리스 노트와 Git 패치, 고객사 OS 이미지까지 다시 확인하며 오탐·미탐의 원인을 찾았습니다.

정합성 문제는 수집 로직만 고쳐서는 잡히지 않았습니다. 같은 컴포넌트가 대소문자 차이로 다른 데이터가 되는 문제는 식별자 정책과 스키마의 문제였고, 파싱 버그를 발견해도 되돌릴 수 없는 문제는 원본을 남기지 않는 저장 구조의 문제였습니다. 그래서 문제가 나온 자리마다 테이블 구조, 저장 구조, 모니터링을 직접 고쳤고, 검토한 방안들과 선택 이유는 비교 문서와 Runbook으로 남겨 다음 사람이 같은 문제를 다시 겪지 않게 했습니다.

경력

래브라도랩스 Data Engineer

2021.06 - Present

소프트웨어 공급망 보안 데이터 플랫폼의 수집·가공·매핑·검증 파이프라인과 공통 실행 환경을 담당합니다. 데이터가 먼저 준비돼야 분석 엔진이 제대로 동작하는 구조라, 데이터팀과 엔진팀이 한 팀처럼 움직입니다. 시스템 전반의 변경은 비교 기준과 방안을 제가 여러 개 만들어 올리고, 팀장·CTO 리뷰를 검증 장치로 삼아 진행했습니다. 개별 크롤러 로직은 각 담당자가, 데이터 모델·공통 실행 환경·34억 행 테이블을 포함한 수십억 행 규모의 DB·스토리지·모니터링은 제가 맡았습니다.

- CVE·OSV와 **12종 OS 배포판**의 보안 권고안을 제품·버전 범위에 매핑했습니다. 배포판마다 제공 방식(API·OVAL·Security Tracker·Git)이 달라 파서를 소스별로 재설계했고, 패치 버전까지 정확히 수집했습니다. [상세]
- 고객사 중앙 운영 DB의 CPU 이상 부하를 실행계획으로 좁혀, collation 불일치로 뷰 조인이 **약 130만 건**을 스캔하던 것(직접 조인은 **약 6,000건**)을 찾아 **스키마·인덱스·식별자 정책을 재설계**해 부하 원인을 구조에서 제거했습니다(사례 1). [상세]
- 수집 DB는 크롤러 쿼리 전수 조사로 미사용 인덱스와 레거시 테이블을 걷어내 **9.6TB → 4TB**로, 배포 DB는 분석 엔진 쿼리 95개를 EXPLAIN으로 전수 분석해 **인덱스 311개 중 224개(72%) 미사용**을 확인·정리해 **2.6TB → 1.2TB**로 줄였고, 고객사 온프레미스 초기 구축이 **12시간 → 6시간**이 됐습니다. [상세]
- 재수집 때마다 **1,000만~1억 건**의 외부 API 요청이 필요하던 원테이크 수집을 RAW 보존·재파싱(ELT) 구조로 재설계했습니다. 적용한 소스부터는 재처리에 외부 요청이 들지 않고, version_id 이력으로 이상 데이터를 재현할 수 있습니다. 현재 단계적으로 적용 중입니다. [상세]
- 수집 주기는 데이터의 시급성으로 정했습니다. 취약점 정보는 고객사가 빨리 대비할수록 가치가 있어 **20분 주기**로 수집하고 1시간 배치와 맞물려 **약 1시간 안에 고객사까지 도달**시켰고, 라이브러리 버전은 새로 나와도 기업이 바로 코드를 고치지는 않으므로 4시간 주기로 늦췄습니다. 이 스케줄을 Airflow·KubernetesExecutor로 **80개+ 크롤러**에 나눠 실행하고, pool·backoff로 외부 API rate limit과 동시 실행 수를 관리했습니다. [상세]
- 크롤러의 DB 직접 연결을 bounded queue와 connection pool을 둔 DML Broker로 바꾸고 Kubernetes 노드마다 Broker pod를 배치했습니다. lock·connection 오류가 하루 3~4건에서 **적용 후 4개월간 0건**입니다. [상세]
- AWS·범용 PC에서 IDC로 대규모 이관을 수행하며, 수집·배포·운영이 인스턴스 하나에 통으로 있던 DB를 셋으로 분리해 부하 간섭을 없앴습니다. 월 10~20건이던 DB 운영 이슈를 수집 스케줄 분산으로 월 5건 미만까지 낮췄고, IDC 이전(2024) 후로는 **1~2건**에 그쳤습니다. 이관을 마친 뒤 운영 DB만 AWS에 남기고 나머지 자원을 정리해 **월 AWS 비용을 약 94%** 줄였습니다. [상세]
- Prometheus·Grafana로 서버·DB replica 상태와 함께 **수집 누락·수집량, 고객사 데이터 동기화 상태**를 같은 시간축에서 보도록 구성했습니다. 토큰·키 만료나 백업 완료 같은 이벤트에는 메신저 알림을 붙여, 사람이 대시보드를 보기 전에 알 수 있게 했습니다. [상세]

문제 해결 방식

장애가 나면 실패한 작업을 곧바로 다시 돌리기보다 **데이터가 어디까지 들어왔는지부터** 확인합니다. Airflow 실행 기록과 Kubernetes·DB 지표를 보고, 필요하면 수집 DB와 배포 DB를 원본 데이터와 직접 맞춰 봅니다. 막힌 지점을 찾으면 서비스부터 정상화하고, 같은 문제가 반복되지 않도록 재시도 방식과 모니터링을 손봅니다. 조치 후에는 오류 건수와 누락 데이터가 실제로 줄었는지 다시 확인합니다.

기술 문제 개선 사례 1 — 라이브러리 컴포넌트 테이블 재설계

중앙 운영 DB CPU 부하의 원인이 된 데이터 모델·식별자 문제

상황 고객사 환경의 중앙 운영 DB에서 CPU 이상 부하가 관측됐습니다. 문제의 테이블은 여러 패키지 생태계(npm·PyPI·Maven 등)의 컴포넌트 정보를 담아 제품의 여러 기능이 기대는 중심 테이블이었습니다.

확인 서버 증설로 덮지 않고 슬로우 쿼리에서 시작해 실행계획으로 좁혔습니다. 같은 값을 저장하는 두 테이블의 collation이 달라(한쪽만 대소문자 구별) 뷰를 통한 조인은 인덱스를 타지 못하고 **약 130만 건**을 스캔하는데, 테이블 직접 조인은 인덱스를 타고 **약 6,000건**에 그쳤습니다. 이어 서로 다른 패키지가 같은 키로 충돌하거나 대소문자 차이로 동일 컴포넌트가 누락되는 식별자 문제를 확인했습니다.

조치 어긋나 있던 collation을 정리하고 **스키마·인덱스·식별자 정책을 재설계**했습니다. 여러 기능이 기대는 중심 테이블이라 당장의 튜닝보다 데이터 모델을 고치는 쪽이 오래 가는 해결이라고 판단했습니다.

결과 CPU 부하의 원인을 구조에서 제거했고, 식별자 충돌·누락이 줄어 정합성이 개선됐습니다. 원인 추적과 구현은 제가 진행했고, 여러 기능이 기대는 중심 테이블이라 재설계는 여러 안을 만들어 비교한 뒤 리뷰를 거쳐 확정했습니다.

상세: portfolio.hwlabs.dev/items/library-table-redesign.html

기술 문제 개선 사례 2 — 서비스 중단 없이 AWS·범용 PC에서 IDC로 이관

발열과 수집 집중으로 DB 서버가 멈추고 데이터가 누락되던 문제

상황 수집·배포·운영(prod)이 AWS의 DB 인스턴스 하나에 통으로 들어 있었고, 비용 문제로 그 구성 그대로 사내 범용 PC로 옮겼습니다. 크롤러가 몰리는 시간대에 DB 쓰기와 I/O가 집중되면 M.2 SSD 발열로 서버가 멈췄고, **한 달에 10~20번** 운영 이슈가 생겨 그대로 데이터 누락과 지연으로 이어졌습니다.

확인 syslog, 장애 시간대의 수집 스케줄, 네트워크 상태를 함께 확인해 특정 시간대의 부하 집중이 프리징과 맞물린다는 점을 찾았습니다.

조치 당장 들일 장비가 없어 먼저 수집 시간을 나눠 이슈를 월 5건 미만으로 낮췄고, IDC 서버가 들어올 때까지 그 상태로 운영했습니다. 서버가 들어온 뒤 이관하면서, 인스턴스 하나가 다 감당하던 구조를 **수집 DB·배포 DB·운영 DB로 분리**해 수집 부하가 배포·운영 조회를 방해하지 않게 만들었고, DB primary/replica, 수집 서버, 백업·복구와 모니터링을 다시 구성했고, 이관 전후 테이블 건수를 대조해 데이터 정합성을 확인한 뒤 전환했습니다.

결과 IDC 전환(2024) 후 DB 운영 이슈는 **1~2건**에 그쳤고, 수시 모니터링으로 서비스 상태를 계속 확인하고 있습니다. 이관을 마친 뒤에는 운영(prod) DB만 AWS에 남기고 나머지 자원을 정리해, **월 AWS 비용을 약 94%** 줄였습니다.

상세: portfolio.hwlabs.dev/items/aws-to-idc-migration.html

그 밖의 운영 개선

MySQL 8.4 LTS 무중단 업그레이드

primary→replica→고객사 on-premise 약 10곳으로 이어지는 복제 체인을 검증, 백업, 단계 롤아웃 순서로 **서비스 중단 없이** 8.0에서 8.4 LTS로 올렸습니다.

백업 자동화

매월 담당자가 매뉴얼을 따라 직접 하던 물리 백업(압축 생성, 용량 단위 분할, 체크섬, NAS 보관)의 전 과정을 메신저 알림까지 이어지는 자동 작업으로 바꿨습니다. **767GB** MySQL 백업을 200GB 단위로 나눠 운영했습니다.

AI 리뷰 CI/CD kit

Jira 브랜치, 테스트, PR 생성, **Claude 기반 PR 리뷰**, Slack 알림, Docker push로 이어지는 개발 완료 루틴을 중앙 kit로 만들어 여러 repo에 같은 방식으로 적용했습니다. diff secret scan을 통과한 뒤에만 외부 모델을 호출합니다.

실행 환경 확장

IDC Kubernetes 2대에 사내 유휴 서버 4대를 WireGuard로 연결해 6대로 늘리고, Pod 할당 실패를 **주 70~100건에서 주 1~5건**으로 줄였습니다.

기술

Data Pipeline: Python, SQL, Apache Airflow, KubernetesExecutor, ETL/ELT, Crawler, Object Storage(SeaweedFS)

Database: MySQL, PostgreSQL(Airflow 메타 DB·RAW 메타정보 저장), EXPLAIN·Index Tuning, Replication, Binary Log, TimescaleDB(동기화 모니터링 지표 저장)

Infrastructure: Linux, Kubernetes, Docker, Helm, IDC/On-premise, AWS(EC2·S3·EFS 운영 후 IDC 이전), NFS

Observability / Automation: Prometheus, Grafana, Shell Script, Go(데이터 전송 모듈 Python→Go 재작성), Jenkins(크롤러 배포), Bitbucket Pipelines

AI Tooling: Claude 기반 PR 리뷰가 포함된 CI/CD 중앙 kit 설계·구현, LLM 도구(Claude Code·Codex) 기반 문서화·운영 자동화

연구 및 학력

세종대학교 시스템보안 연구소 연구원

2018.07 - 2021.02

Linux 임베디드 환경에서 UART 펌웨어 획득, QEMU 에뮬레이션, AFL 계열 퍼징과 심볼릭·콘콜릭 실행을 사용해 IoT 펌웨어의 실행 경로와 취약점을 분석했습니다. 석사학위 연구를 확장한 FIRM-COV·EF-Fuzz를 SCI급 국제 저널 논문으로 정리했습니다.

세종대학교 대학원 · 정보보안 석사

2018.09 - 2021.02

지원서 — Why Remember

왜 이 직무인가

지난 5년간 제 일의 대부분은 서로 다르게 표기된 데이터를 같은 것으로 식별하고 연결하는 일이었습니다. 생태계마다 다른 패키지 표기, 12종 배포판의 보안 권고안, 대소문자 차이로 갈라지던 식별자 — 형태는 달랐지만 전부 **이게 같은 것인가**를 데이터 구조로 답하는 문제였습니다.

데이터 파운데이션팀 공고에서 **정교한 매핑 시스템과 마스터 데이터 관리**, **타협할 수 없는 데이터 정합성**이라는 문장을 읽고 지원을 결정했습니다. 명함과 프로필도 같은 사람과 같은 회사가 여러 표기로 들어오는 데이터입니다. 그 식별과 매핑의 정확도가 서비스 신뢰를 결정한다는 점에서, 제가 취약점 데이터에서 다뤄온 문제와 구조가 같습니다.

왜 리멤버인가

리멤버는 주변 추천으로 명함을 등록하면서 처음 썼습니다. 명함을 관리하는 앱이라고만 생각했는데, 프로필을 올려 두니 스카우트 제안이 오고 지원할 수 있는 공고가 쌓였습니다. 제안을 받아 보며 프로필과 공고가 실제로 연결된다는 것을 사용자로서 확인했습니다. 그리고 그 연결을 만드는 데이터 파운데이션팀의 공고를 보고, 제안을 기다리는 대신 직접 지원서를 썼습니다.

사용자로 쓰다 보니 이 서비스의 데이터가 궁금해졌습니다. 명함 한 장에도 같은 회사가 (주)OO와 OO 주식회사처럼 갈라져 들어올 텐데, 이 식별과 매핑이 먼저 맞아야 **이 프로필에 이 공고**라는 연결이 성립합니다. 명함과 프로필은 개인정보라 접근 통제와 보존 정책이 정합성만큼 설계의 제약이 된다는 점도, 보안 데이터를 다루며 몸에 익은 감각과 이어져 있습니다. 지금까지 보안 데이터에서 풀던 식별·매핑 문제를, 사람과 기회를 연결하는 데이터에서 더 단단하게 풀어 보고 싶습니다.

무엇을 기여할 수 있는가

정합성 문제를 만나면 담당자의 주의력에 기대는 대신 구조를 고쳐 왔습니다. 이 방식은 명함·프로필 데이터에서도 그대로 쓰일 수 있습니다. 원본(RAW)을 보존해 재파싱하는 구조는 매핑 로직을 고친 뒤 과거 데이터를 다시 세울 수 있는 토대가 되고, 수집 누락·동기화 상태를 지표로 만들어 상시 관측해온 방식은 데이터 품질 지표 정의와 이상 감지로 이어집니다. 검토한 방안과 선택 이유를 비교 문서와 Runbook으로 남기는 습관은 팀의 자산으로 가져가겠습니다.

공고의 **반복적인 운영 업무를 기술로 자동화**도 이미 해오던 방식입니다. 매월 수작업이던 물리 백업을 압축·분할·체크섬·보관·메신저 알림까지 전 과정 자동화했고, 최근에는 LLM 도구를 문서화·리뷰·반복 운영에 쓰고 있습니다. 표기가 갈라진 데이터를 같은 것으로 잇는 일을, 리멤버의 데이터에서 계속하고 싶습니다.