LLM 기반 SAST 스캐너 구축기

"지속 가능한" 보안 코드 리뷰 자동화 전략 및 구현

힐링페이퍼(강남언니) / Security Engineer / 한종훈

Agenda

Intro & Problem Definition

Speaker

현) 힐링페이퍼(강남언니) - Security Engineer / 한종훈

전) 스틸리언 - R&D | 모의해킹, CTF 운영/솔루션

취미 : 클라이밍, CTF

더 좋은 의료 서비스를
누구나 누릴 수 있게

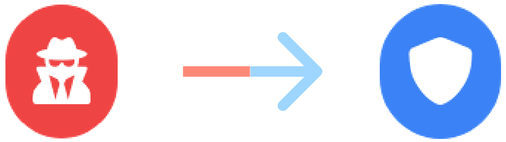
HEALING PAPER

강남언니 カンナムオンニ UNNI

임직원 : 300명
누적 사용자 : 830만명

힐링페이퍼는 공급자 중심의 의료 서비스를 IT 기술로 혁신하여 고객 중심으로 전환할 것을 추구합니다

Intro

Think like an attacker,  build like a defender.

공격자 관점에서는 취약점 하나만 찾아도 기뻤지만,
보안 담당자가 되고 보니 매일 쏟아지는 보안 알림과 식별되지 않는 취약점에 대한 공포로 고통 받고 있습니다.

Goal & No Goal

SAST와 LLM을 실제 CI 파이프라인에 붙여본 아키텍처 공유

왜 상용툴 대신 Semgrep + LLM을 직접 구축하게 되었는지 의사결정 과정

개발자 경험을 해치지 않는 보안 코드 리뷰 운영 고민

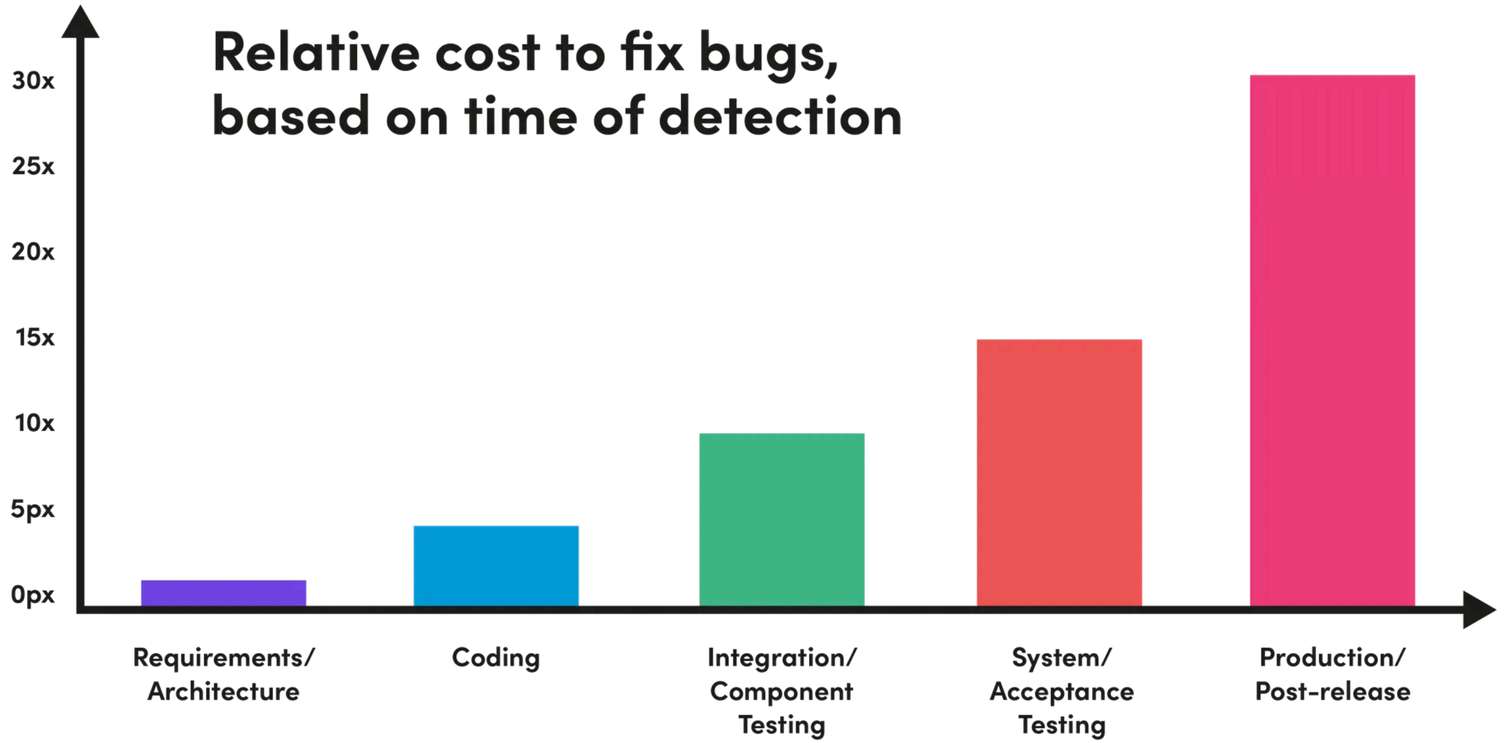
LLM 모델 성능 벤치마크

모델 비용 최적화 팁

특정 소프트웨어 평가 및 추천

완벽한 정답 제시

왜 우리의 SAST는 지속 가능하지 않은가?



이미지 출처 : https://mondoo.com/blog/should-your-infrastructure-security-shift-left-or-right

왜 우리의 SAST는 지속 가능하지 않은가?

The Noise

보안 경보의 **53%** 절반 이상이 오탐(False Positive)

개발자가 오탐 1건을 판별하는 데 **15-30분** 소요

55%의 조직이 마감 압박으로 보안 스캔 건너뜀

→

The Fatigue & Untrust

보안팀 신뢰도 하락

개발팀의 부담 증가

의도한 방향과 다른 DevSecOps 환경

참고. The State Of DevSecOps Report | Black Duck Global State of DevSecOps 2024

왜 우리의 SAST는 지속 가능하지 않은가?

취약점 유형	영향도	우선순위	조치 난이도	공격 벡터	발견 건수	확인된 위험	발생 가능성	검증 방식
SQL Injection - hibernate-sql - sqlalchemy.text	High	P0	낮음	Raw query 및 사용자 입력 처리 API	13건	공격 가능한 엔드포인트 API 확인	High	PoC 공격 테스트, 코드 분석
Improper Authorization - missing-user - missing-user-entrypoint	Medium	P1	중간	취약한 인터페이스 설정 및 권한 상승 위험 취약한 인터페이스 이름지	4개건	Root 실행 인터페이스에서 호스트 접근 가능	Medium	관련 CVE 확인 및 코드 분석
Cross-Site Scripting - inner-html - react-dangerouslysetinnerhtml	Medium	P1	낮음	사용자 웹 입력 값	5건	세션쿠키 탈취 및 악성 리다이렉션 가능	Medium	코드 분석
Improper Validation - CRLF-injection-logs	Low	P2	낮음	사용자 입력 기반 로그 인젝션	12개	로그 변조 및 정보 노출 가능	High	코드 분석
Active Debug Code - active-debug-code-printstacktrace	Low	P2	중간	사용자 입력에 대한 예외처리	약 40건	일부 시스템 정보 노출(코드 상세 에러, 디버깅 에러 등)	Medium	PoC 공격 테스트, 코드 분석

Critical 영역에서 오탐율 20% 미만
High/Medium 영역에서 오탐율은 50% 이상

왜 우리의 SAST는 지속 가능하지 않은가?


```
❯❯ javascript.vue.security.audit.xss.templates.avoid-v-html.avoid-v-html
       Dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily
       lead to XSS vulnerabilities. Only use HTML interpolation on trusted content and never on user-
       provided content.
```

취약점 백과사전

> 참고. 해당 취약점 백과사전은 LLM에 의해 생성되었습니다. 다소 잘못된 정보를 전달할 수 있습니다.

▶ 사용된 프롬프트

왜 우리의 SAST는
지속 가능하지 않은가?

Critical/High 등급만 리뷰해도 1–2주가 걸립니다.
이 결과를 그대로 개발자에게 넘기면,
의미(Semantic)와 맥락(Context)이 부족해서 실제 수정까지 이어지기 어렵습니다.

문화가 자리잡기 위해서는 시간이 필요하였고
매일 수십게 배포가 이루어지는 환경에서 의미 있는 자동화가 필요했다

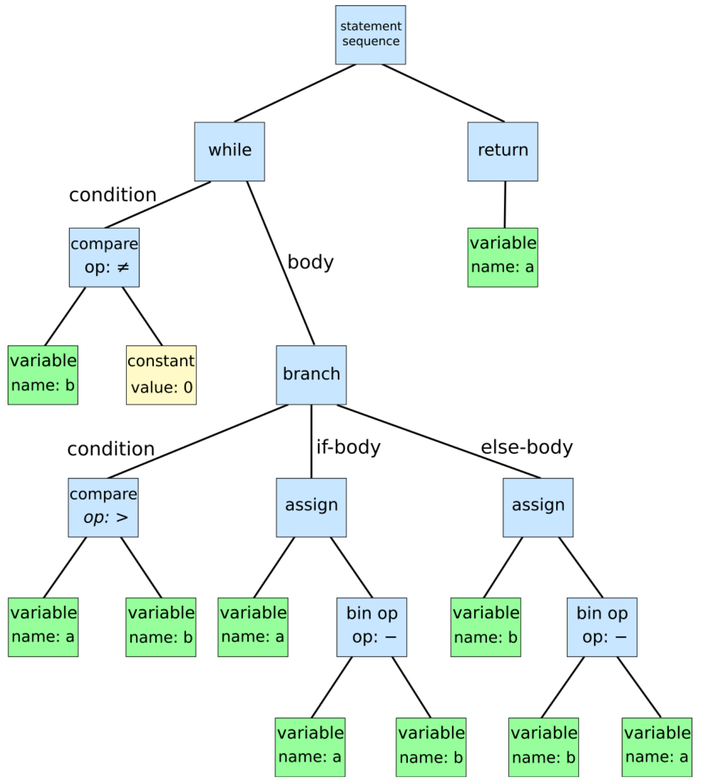
Strategy & Why
왜 SAST + LLM인가?

SAST

코드를 실행하지 않고 정적으로 취약점을 찾는 테스트 방법
SAST는 코드를 실행하지 않고 애플리케이션의 소스 코드, 바이트 코드 또는 컴파일된 버전을 분석

구성 요소

AST + Pattern + Analysis Engine.



Data Flow

Control Flow

Taint

Semantic

기존 SAST의 기술적 한계: '맥락(Context)의 부재'

기존 SAST 대부분 Syntax(모양) 기반 취약점 취약점을 찾기에 특화됨

AST(Abstract Syntax Tree)

전통적인 SAST는 코드를 AST(추상 구문 트리)라는 구문 트리로 파싱(parsing)하여 분석

Pattern Matching

패턴 매칭은 구문 분석을 통해 얻은 코드 구조를 사전에 정의된 위험 패턴 목록과 비교하는 과정

Analysis Engine

AST 기반 Dataflow, Taint 등과 같이 Source(오염원)와 Sink(코드 지점) 데이터 흐름 분석

```java
// File 1: Public API Endpoint
@RequestMapping("/user/{id}")
public User getUserData(@PathVariable String id) {
    // ------------------------------------------------------
    // [Source] 사용자 입력 (id)
    // ------------------------------------------------------
    String query = "SELECT name, email FROM users WHERE id = '" + id + "'";

    return DatabaseService.executeQuery(query); // [Sink]
}


// File 2: DatabaseService.java (별도의 안전한 유틸리티 클래스)
class DatabaseService {
    // 내부적으로 Prepared Statements를 사용하거나 ORM을 사용하는 로직
    public static User executeQuery(String unsafeQuery) {
        // *** 실제로는 여기서 안전하게 파라미터 바인딩 처리됨 ***
        PreparedStatement safeStmt = connection.prepareStatement(unsafeQuery);
        // ... 안전한 실행 로직 ...
    }
}
```

Case - SQL injection

Dataflow 와 같은기술을 통해 preparedStatement/executeQuery/ parameterized Bind 호출 여부를 찾을 수 있음.

단, 사전에 룰이 정의되어 있어야함.

1. DatabaseSevice
2. executeQuery
3. preparedStatement
4. parameterized Bind

커버리지를 위한 패턴에 대한 사전 정의

```java
public class AuthenticationService {

    private Map<String, UserSession> activeSessions = new ConcurrentHashMap<>();
    private UserRepository userRepository;
    /* ... */
    public UserProfile getUserProfile(String sessionToken, String userId) {
        UserSession session = activeSessions.get(sessionToken);

        User user = userRepository.findById(userId);

        return new UserProfile(
            user.getId(),
            user.getEmail(),
            user.getPersonalData()
        );
    }
}
interface UserRepository {
    User findByUsername(String username);
    User findById(String id);
    void delete(String id);
    void updateRole(String id, String role);
}
```

Case - Information disclose/expose

인증 로직/정보 노출 취약점을
SAST가 찾으려면
어떤 패턴을 정의해야할까?

패턴 정의에 대한 부담감

기존 SAST의 기술적 한계: '맥락(Context)의 부재'

그냥 SAST도 있으면 좋긴하겠지만 욕심 좀 내볼게요!

실제 우리가 원하는 것

- **Semantic(의미) + Context(맥락) 기반한 취약점 찾기**
- **세부적인 패턴 관리 하지 않기**
- **하나 알려주면 찰떡같이 알아서 하기**
- **취약점에 대한 오탐 판단, 근거, 대응 방법 전달하기**
  - **신뢰할 수 있는 리뷰어**

SAST + LLM

Context 활용 한계를 충분한 Context를 제공하여 LLM으로 해결

```
❯❯ javascript.vue.security.audit.xss.templates.avoid-v-html.avoid-v-html
      Dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily
      lead to XSS vulnerabilities. Only use HTML interpolation on trusted content and never on user-
      provided content.
```

vs.

🔍 **문제점**: execute 메서드에서 TOCTOU(Time-of-Check-Time-of-Use) 경쟁 조건 취약점이 존재합니다. 라인 25에서 process.isExpired() 체크와 라인 26에서 실제 expire() 처리 사이에 시간 간격이 있어, 동시성 환경에서 다른 스레드가 해당 프로세스의 상태를 변경할 수 있습니다. 특히 포인트 리워드 시스템에서는 금전적 가치가 있는 포인트를 다루므로, 경쟁 조건으로 인해 만료되어야 할 프로세스가 처리되거나 중복 처리될 위험이 있습니다. 또한 라인 21-23의 null 체크와 실제 처리 사이에도 데이터 변경 가능성이 존재합니다.

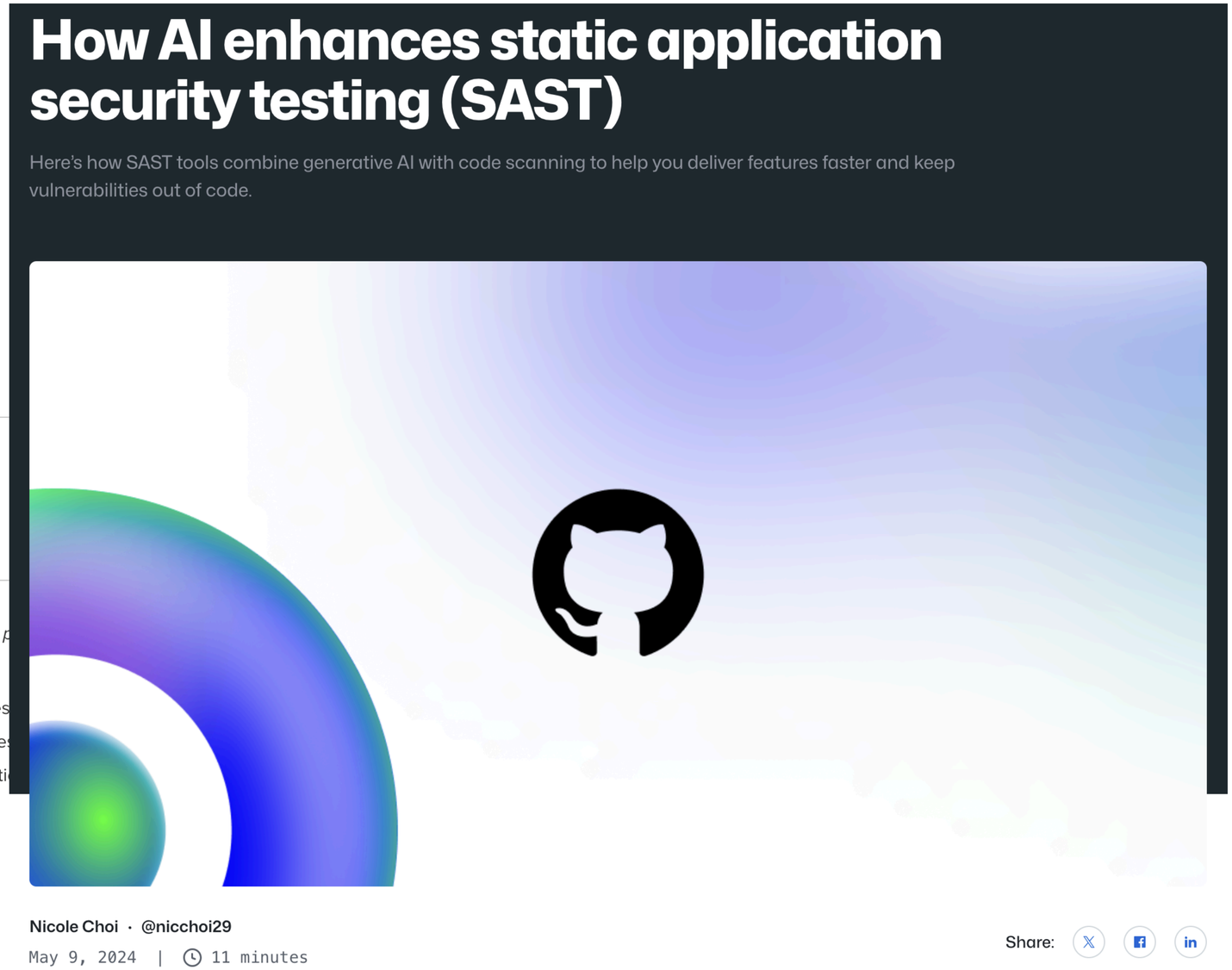
시장의 시그널

Semgrep Assistant Enters General Availability, using AI to 10x the Productivity of Any AppSec Team

NEWS PROVIDED BY
Semgrep →
Mar 20, 2024, 09:00 ET

Semgrep Assistant turbocharges AppSec workflows, streamlining pre and post-scan p

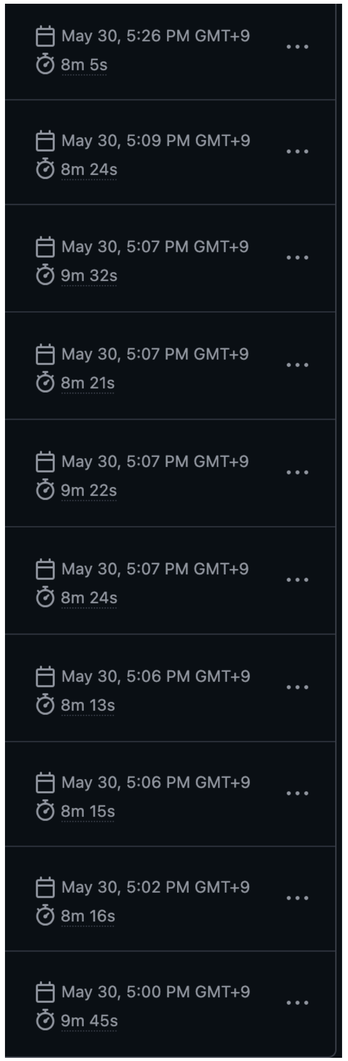
SAN FRANCISCO, March 20, 2024 /PRNewswire/ -- **Semgrep**, a code security solution des
programs, today announced the General Availability of **Semgrep Assistant**, a tool that use
efficiencies and uncover insights across all phases of an AppSec program, from rule creati



Build vs. Buy

마주친 현실

- 유료 도구 쓰면 ROI 나오려나?
- 이거 쓰면 원하는 결과 나올까? Remediation 아쉽네.
- PoC 를 하긴했는데 프로세스도 제대로 없네?
- 생각보다 많이 비싼 거 같은데?
- 소스코드 저기 넘겨도 괜찮은가?
- 생각보다 스캔 너무 오래 걸리는데?



Build vs. Buy

마주친 현실

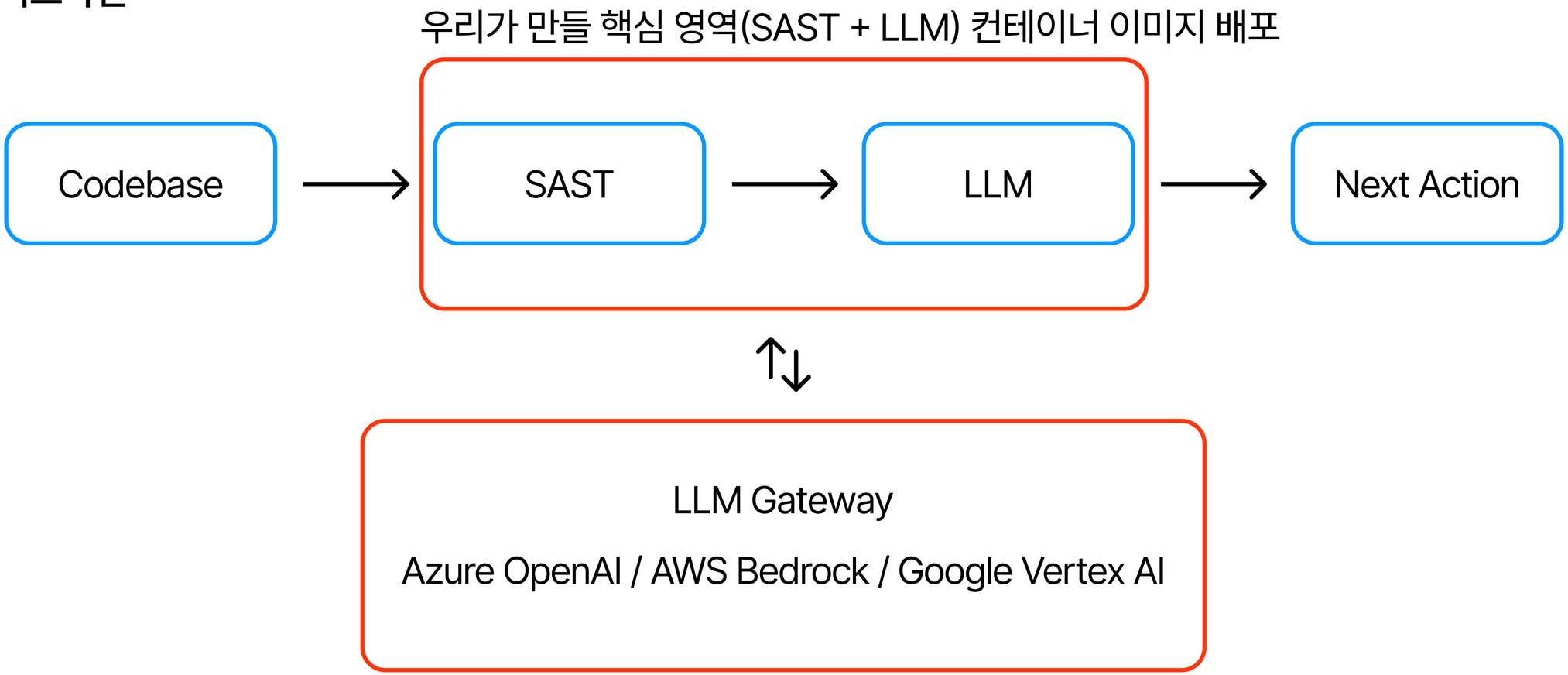
- 유료 도구 쓰면 ROI 나오려나?
- 이거 쓰면 원하는 결과 나올까? Remediation 아쉽네.
- PoC 를 하긴했는데 프로세스도 제대로 없네?
- 생각보다 많이 비싼 거 같은데?
- 소스코드 저기 넘겨도 괜찮은가?
- 생각보다 스캔 너무 오래 걸리는데?

확신 없으니 직접 Build and Small Start 해보자

Engineering Journey
실패와 성공의 기록

야심 찼던 실패: 복잡한 AI 에이전트 접근

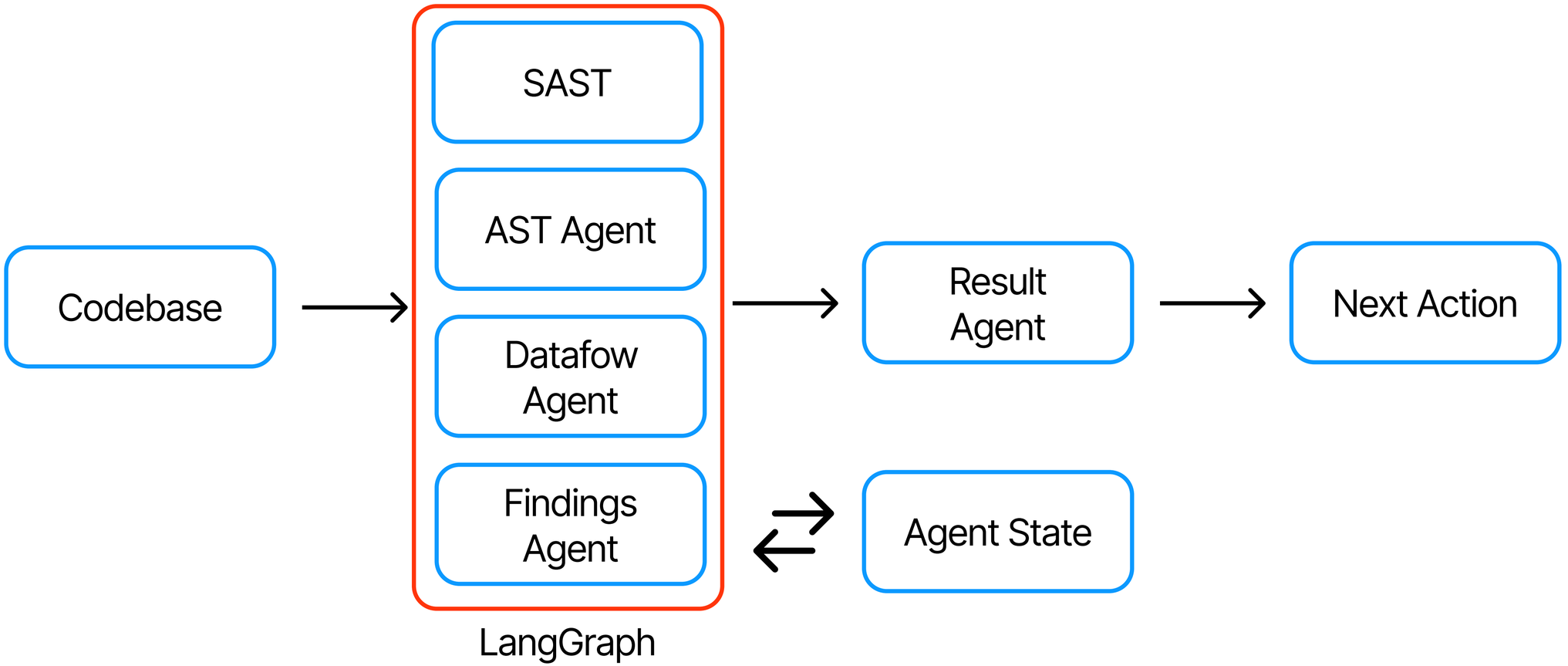
공통 파이프라인



야심 찼던 실패: 복잡한 AI 에이전트 접근

멀티 에이전트 기반 파이프라인

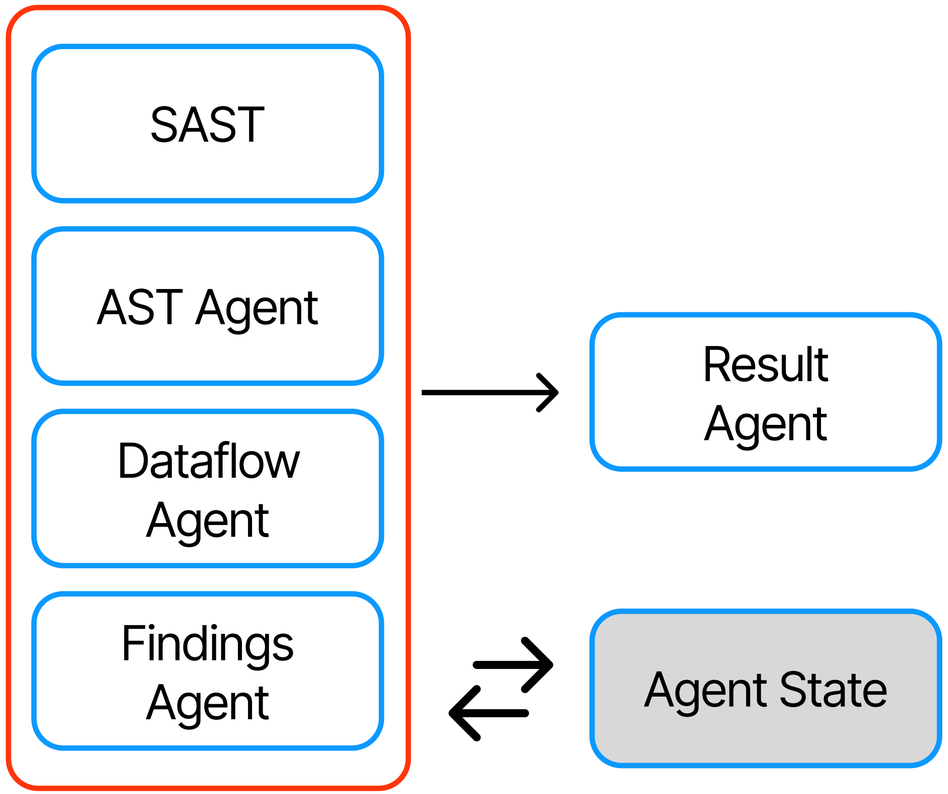
model: gpt 4.1-mini



야심 찼던 실패: 복잡한 AI 에이전트 접근

Agent State
모든 에이전트가 공유하는 작업 메모리

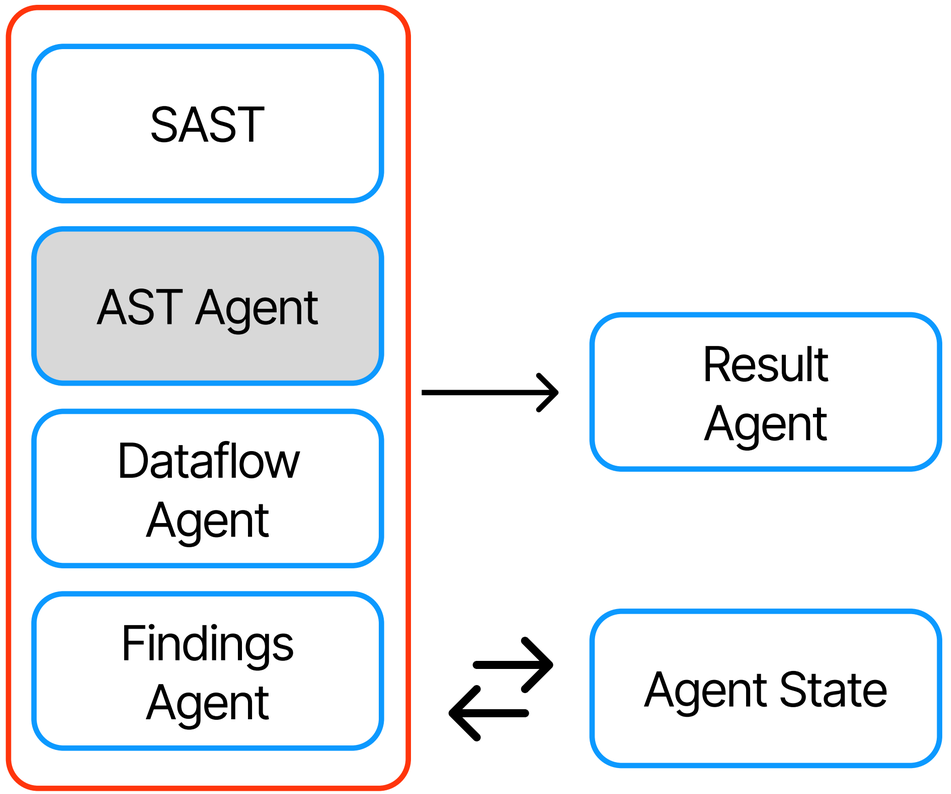
```python
class AgentState(TypedDict):
    file_path: str
    source_code: str
    raw_findings: List[Dict]

    # 각 에이전트가 채워넣을 데이터
    ast_structure: Dict[str, Any]]
    potential_findings: List[Dict]
    verified_flows: List[Dict]
    final_report: str
    errors: List[str]
    ...
```



야심 찼던 실패: 복잡한 AI 에이전트 접근

AST Agent
코드를 읽고 "함수, 클래스, 변수 관계"만 파싱하여 구조화된 데이터 (JSON) 변환

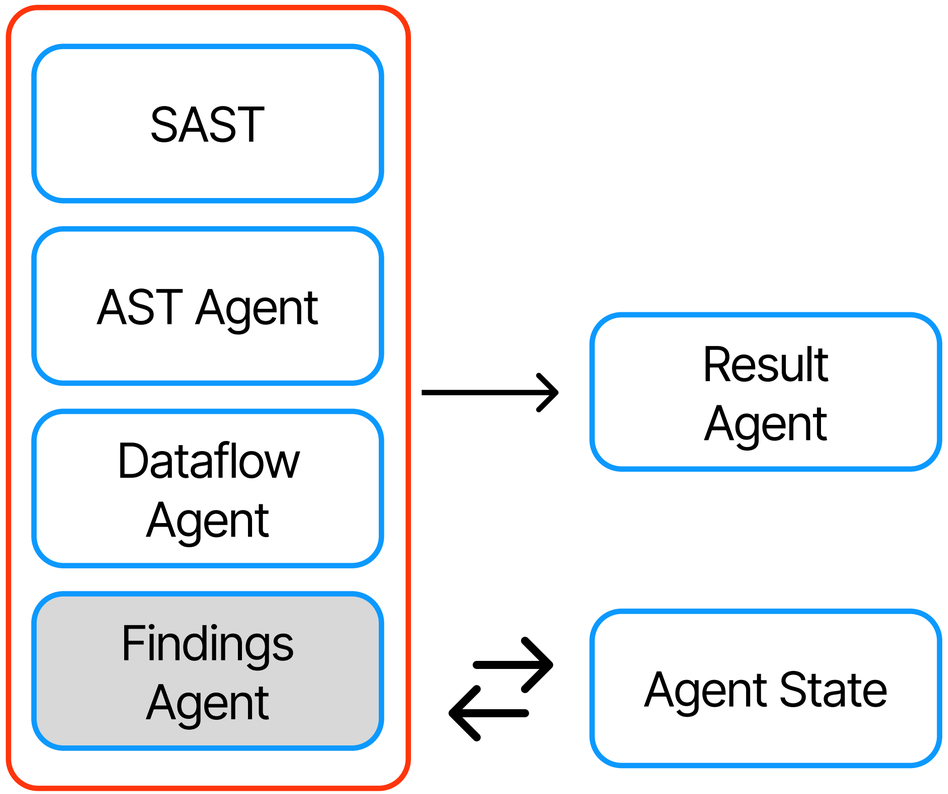
```python
def ast_agent(state: AgentState) -> AgentState:
    prompt = PromptTemplate.from_template("""
    너는 AST 파싱 전문가야. 아래 코드를 분석해서 함수 목록, 주요 클래스, 그리고 Import된 라이브러리를 JSON 포맷으로 추출해줘.
    ###Code:
    {code}
    ###Output: {...JSON Structure}
    """)
    chain = prompt | llm | JsonOutputParser()
    result = chain.invoke({"code":state["source_code"]})
    state["ast_structure"] = result
    return state
```



야심 찼던 실패: 복잡한 AI 에이전트 접근

Findings Agent
AST 정보와 소스 코드를 보고 잠재적인 보안 취약점 후보군(Candidates)을 찾음. 혹은 SAST가 찾은 취약점 오탐 검증

```
1  def findings_agent(state: AgentState) ->
   AgentState:
2      prompt = PromptTemplate.from_template("""
3      너는 세계 최고 소스 코드 보안 분석가야. AST 구조와 코드를
   보고 OWASP Top 10 취약점/비즈니스 로직 취약점을 찾아줘.
4      특히 SQL Injection, XSS, Command Injection, RCE,
   NPE, OOB, IDOR, XXE 위주 ...
5      [AST 구조]: {ast}
6      [소스 코드]: {code}
7      ###반드시 결과(Output)는 JSON 리스트로 반환해: [{{
   "type": "...", "line": 10, "code": "..." }}]
8      """)
```

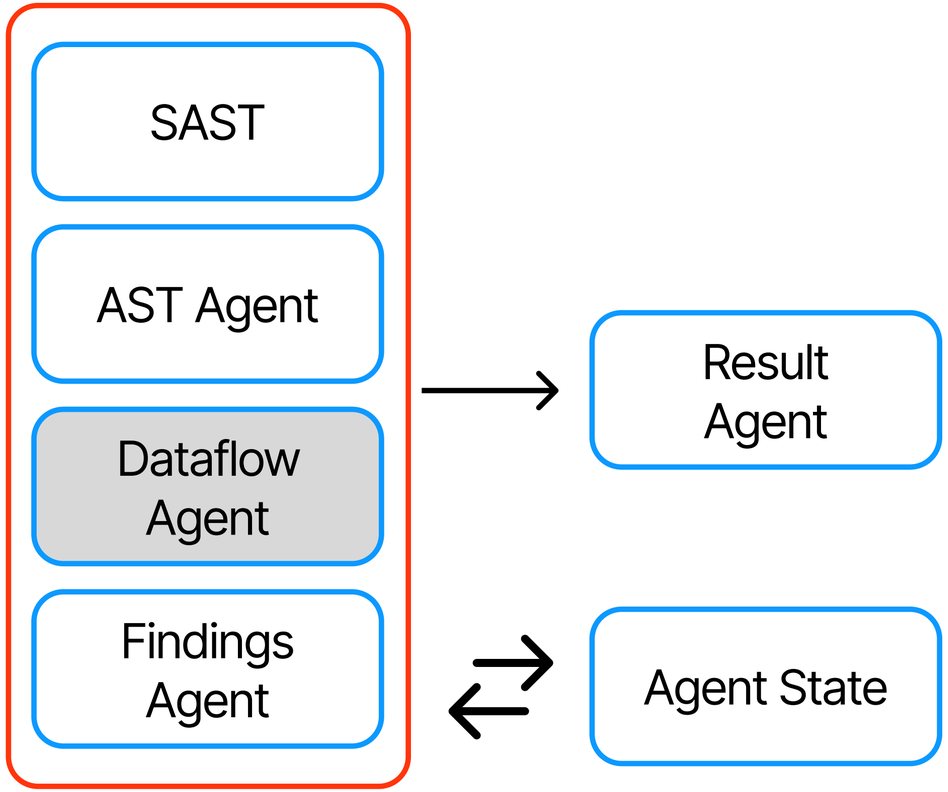


야심 찼던 실패: 복잡한 AI 에이전트 접근

Dataflow Agent

발견된 취약점 후보에 대해 "사용자 입력값(Taint Source)이 정말 여기까지 도달하는가? 를 확인 혹은 AST 단계와 확인

```
def dataflow_agent(state: AgentState) -> AgentState:
    findings = state["potential_findings"]
    verified = []
    prompt = PromptTemplate.from_template("""
        너는 Taint Analysis 전문가야.
        {line}번 라인의 코드가 보안 취약점으로 지목되었어.
        코드: `{vuln_code}`
        이 변수가 '사용자 입력(Request Parameter 등)'으로부터 시작되어
        검증 없이 도달하는지 실제 엔드포인트를 판단해줘.
        전체 코드:
        {full_code}
        결과는 JSON으로: {{ "is_reachable": true/false, "reason": "..." }}
        """)
```



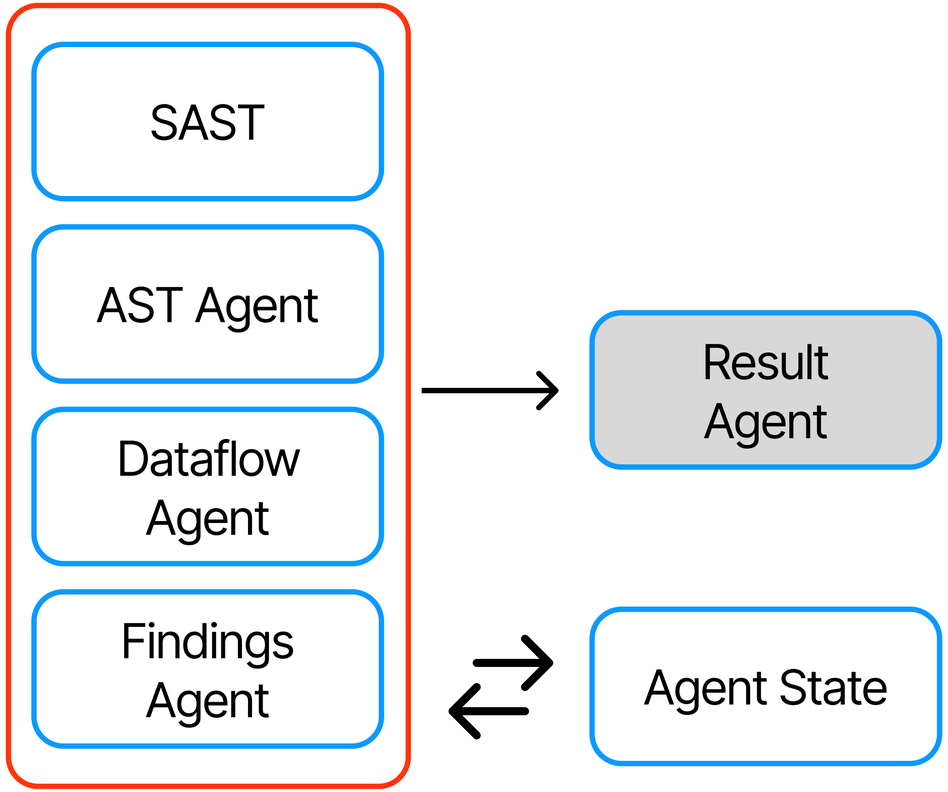
야심 찼던 실패: 복잡한 AI 에이전트 접근

Result Agent
검증된 정보를 종합하여 개발자가 이해하기 쉬운 최종 보고서를 작성

```
def result_agent(state: ScanState) -> ScanState:
    prompt = PromptTemplate.from_template("""
    너는 보안 리포트 작성자야. 검증된 취약점 정보를 바탕으로
    개발자가 수정할 수 있는 Markdown 리포트를 작성해줘.

    [검증된 취약점 목록]:
    {verified}

    포맷:
    ## 🚨 발견된 취약점
    - 유형: ...
    - 위치: ...
    - 원인: (Dataflow 분석 결과 인용)
    - 수정 제안: ...
    """)
```



와 대박날 거 같다.

나름 잘 동작하는거 같은데?

▼ 🤖 AI 추가 발견 - 기타 (4개) - 클릭하여 펼치기

1. 🎯 `code-security/vulnerable_samples/VulnerableBankAPI.java:15`

- **문제:** 하드코딩된 비밀 API 키와 데이터베이스 연결 정보가 소스 코드에 직접 포함되어 있습니다. 이는 심각한 보안 위험을 초래할 수 있습니다. 공격자가 소스 코드에 접근하면 이 중요한 정보를 쉽게 탈취할 수 있습니다.
- **수정 제안:**

```
// 환경 변수나 안전한 비밀 관리 시스템을 사용하여 중요 정보 관리
private static final String THIRD_PARTY_API_KEY = System.getenv("THIRD_PARTY_API_KEY");

// 데이터베이스 연결 정보도 환경 변수로 관리
private static final String DB_URL = System.getenv("DB_URL");
private static final String DB_USER = System.getenv("DB_USER");
private static final String DB_PASSWORD = System.getenv("DB_PASSWORD");
```

2. 🎯 `code-security/vulnerable_samples/VulnerableBankAPI.java:156`

- **문제:** 이 엔드포인트는 서버 종료 기능을 제공하지만, 인증이나 권한 검사가 전혀 없습니다. 이는 매우 위험한 보안 취약점으로, 공격자가 무단으로 서버를 종료할 수 있습니다.
- **수정 제안:**

```
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.Authentication;

@PostMapping("/admin/shutdown")
@PreAuthorize("hasRole('ADMIN')") // Spring Security를 사용한 권한 검사
public String shutdownServer(Authentication authentication) {
    // 추가적인 권한 검증
    if (!isAuthorizedAdmin(authentication)) {
        throw new AccessDeniedException("Unauthorized shutdown attempt");
    }

    logEvent("Server shutdown initiated by: " + authentication.getName());
    // 실제 종료 로직
    return "Shutdown signal received. Server is terminating.";
}

private boolean isAuthorizedAdmin(Authentication authentication) {
    // 추가적인 권한 검증 로직 구현
    return true; // 예시 구현
}
```

와 대박날 거 같다.

나름 잘 동작하는거 같은데?

그렇게 약 6주간 문제? 없이 운영되었다.

@here 존경하는 개발자 여러분! 몇주 전 적용했던 **Github Action 내 코드 취약점 찾는 code-security 로직에서 오류 발견해서 수정**하고 있습니다.

- 해당 워크플로우가 실패해도 Mergegate 에서는 실패 처리를 하지 않게 예외처리되어 있어서 머지는 가능합니다. 양해부탁드립니다.

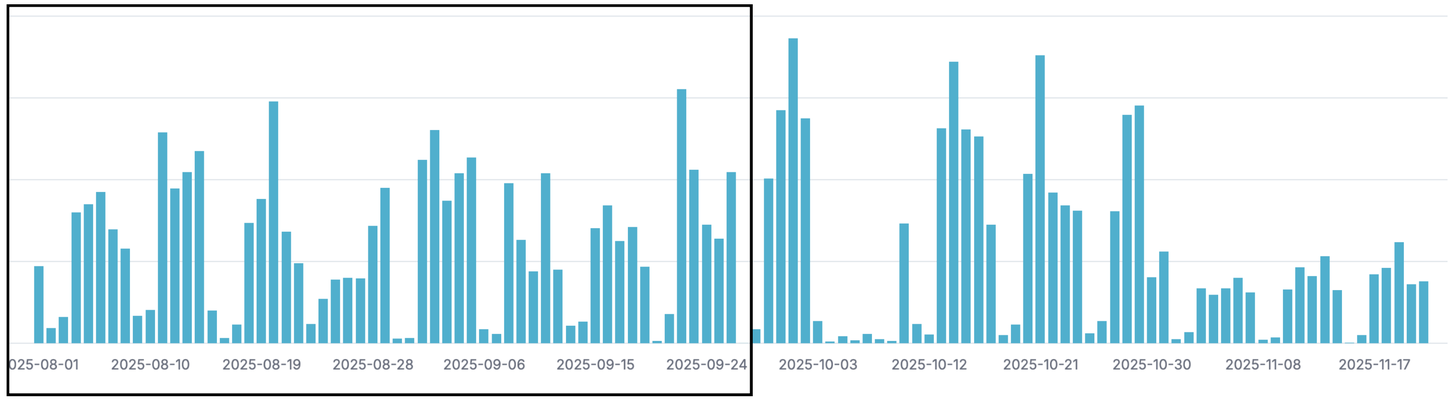
~~와 대박날 거 같다.~~

~~너름 잘 동작하는거 같은데?~~

그렇게 약 6주간 문제? 있는 상태로 운영되었다.

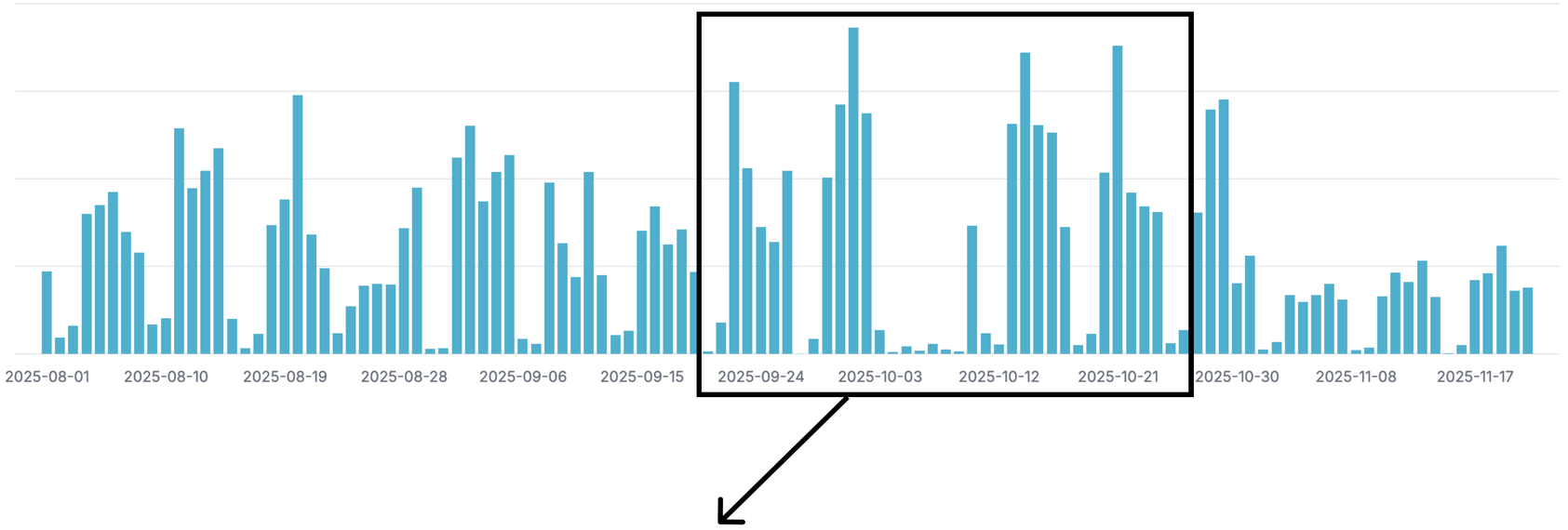
여러 개발자 피드백과 자세한 분석으로 아래 문제 도출!

- 취약점 등급 체계 혼란
- Agent 간 일관성 부재
- 프롬프트 일관성/유지보수 문제
- 비용 문제 (모델 여러 번 호출)
- 속도 문제 (4-step Latency)
- 라인 번호 불일치
- ...

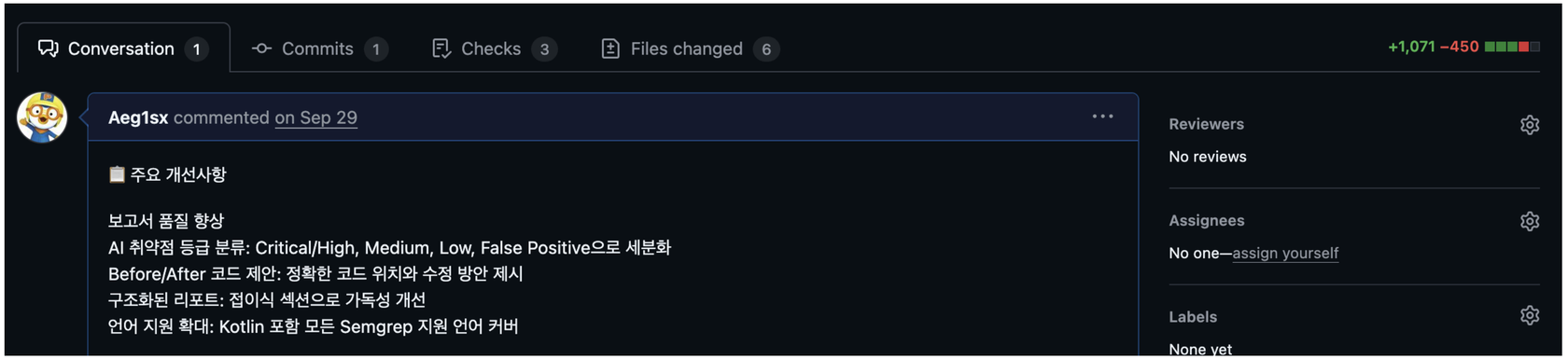


단계별로 전문가(LLM Agent)가 데이터를 가공하고 넘겨주는 것이 이론적으로 완벽했으나 복잡성이 높았다.

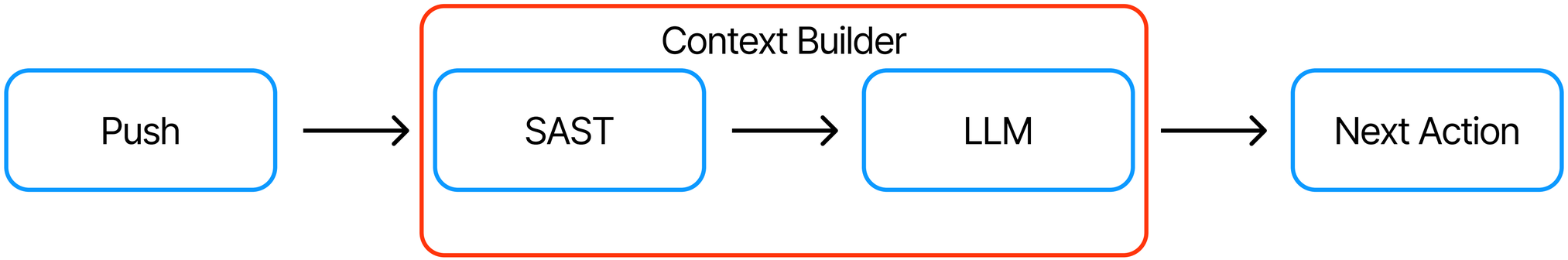
변경 작업



휴 그냥 기존 코드 날려버리자



One-Shot Context Engineering



파이프라인이 복잡할 필요는 없었습니다.
프롬프트는 단일, 추론에 필요한 Context 를 확실히 모아 전달
LLM 내부적으로 CoT 하게 합니다. (Simple is Best)

Ref.
https://github.com/anthropics/claude-code-security-review
LLM-Driven SAST-Genius: A Hybrid Static Analysis Framework for Comprehensive and Actionable Security

One-Shot Context Engineering

핵심은 의미와 맥락

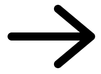
기존 멀티 에이전트

AST Agent

Findings Agent

Dataflow Agent

Result Agent



One-Shot Context

One Agent
One Workflow
One Prompt

문제점 (Pain Points)	해결책 (Solution)
1. 취약점 등급 체계 혼란	Semgrep 기준 통일 + LLM 재조정 Semgrep의 severity를 기준으로 삼고, LLM은 이를 상향/하향 조정(Adjustment)하는 역할만 수행
2. 기존 SAST와 비슷한 문제 Agent도 결국 코드 문맥을 다 못 보고 오탐을 냄	Regex 기반 구조 분석 + Import 추적 단순 텍스트가 아니라 "함수 범위", "Import 관계"를 Regex로 추출하여 LLM에게 맥락(Context)을 강제 주입
3. Agent 간 일관성 부재	Single-Pass Processing 여러 Agent가 따로 놀지 않게, 하나의 LLM 호출 안에서 검증(Verification)과 설명(Explanation)을 동시에 수행
4. 프롬프트 일관성/유지보수 문제	Golden Prompt 가장 잘 동작하는 하나의 거대 프롬프트에 모든 지시사항(Role, Task, Knowledge, Output Format)을 통합 관리
5. 비용 문제 (싼 모델 써도 비쌈)	청크(Chunk) 단위 배치 처리 취약점 하나당 호출이 아니라, 연관된 파일들을 묶어서(Group) 한 번에 여러 취약점을 검사
6. 속도 문제 (4-step Latency)	병렬성 없는 직렬화 제거 Semgrep(매우 빠름) → 전처리(Regex, 빠름) → LLM(1회) 구조로 LLM 대기 시간을 최소화
7. 줄 번호(Line Number) 불일치	원본 코드 맵핑 + Highlight LLM에게 줄 번호를 묻지 않고, 우리가 줄 번호를 매겨서 준 뒤 "이 코드가 맞는지" 확인시킴

How

Golden-Prompt

Role : 시니어 보안 엔지니어, 보안 전문가

Task : 취약점 점검 기준, 오탐 판단, 근거, 수정 제안 등

Knowledge : 코드,사내 환경,아키텍처 일부,공통 모듈 등

```
# 개선된 검증 프롬프트 - Before/After 중심으로 수정
PROMPT_FOR_VERIFICATION = """
당신은 보안 전문가입니다. Semgrep 탐지 결과를 분석하여 실제 취약점인지 오탐인지 판단하고, 실제 취약점의 경우 정확한 수정 방법을 제시하세요.

**분석 기준:**
1. **실제 취약점**:
   - 공격 시나리오와 영향도 분석
   - 취약한 코드의 정확한 라인 범위 식별
   - 실제 실행 가능한 수정 코드 제공 (import 문 포함)
   - 수정 전후 차이점 명확히 설명

2. **오탐 (False Positive)**:
   - 왜 안전한지 구체적 근거 제시
   - remediation_code는 null로 설정

**수정 코드 작성 원칙:**
- 취약한 라인만 수정하되, 필요한 import 문은 포함
- 기존 코드 구조와 스타일 유지
- 실제 동작하는 코드 제공
- 주석으로 수정 이유 간단히 설명

**응답 형식 (JSON):**
```json
{{
  "analyzed_vulnerabilities": [
    {{
      "check_id": "체크ID",
      "path": "파일경로",
      "is_false_positive": false,
      "explanation": "취약점 설명: 공격 방법, 영향도, 위험성",
      "remediation_code": "정확한 수정 코드 (실행 가능한 형태)"
    }}
  ]
}}
```

{format_instructions}

탐지된 취약점:
```json
{semgrep_findings}
```

소스 코드 컨텍스트:
{source_code_map}

중요: remediation_code는 실제 파일에 적용할 수 있는 완전한 코드 블록이어야 합니다. 부분적인 수정이 아닌 전체 함수나 구문 블록을 제공하세요.
"""
```

How

Golden-Prompt

```
**우선 탐지 대상:**
1. **비즈니스 로직 취약점**:
   - 인증/인가 우회 로직
   - 권한 상승 가능성
   - 데이터 접근 제어 허점
   - 결제/거래 로직 조작 가능성

2. **복합 취약점**:
   - 여러 함수에 걸친 데이터 흐름 문제
   - 상태 관리 오류 (race condition, TOCTOU)
   - 세션/캐시 조작 취약점
   - API 엔드포인트 간 로직 불일치

3. **환경/설정 취약점**:
   - 하드코딩된 비밀값
   - 안전하지 않은 기본 설정
   - 에러 처리에서 정보 노출
   - 로깅에서 민감정보 노출

**분석 방법:**
- 코드 흐름을 따라가며 종단간 보안 검토
- 입력 검증과 출력 인코딩 쌍 확인
- 권한 체크와 실제 동작 로직 간 일치성 확인
- 예외 상황에서의 보안 유지 여부 검토

**Severity 판단 기준:**
각 취약점의 위험도를 다음 기준으로 분류하세요:

- **critical**: 즉각적인 시스템 침해 가능
  예: Remote Code Execution, SQL Injection, Authentication Bypass, Account Take

- **high**: 민감 데이터 노출 또는 중요 기능 우회 가능
  예: XSS, CSRF, Path Traversal, Broken Access Control, Session Hijacking, API

- **medium**: 정보 노출 또는 제한적인 보안 영향
  예: Information Disclosure, Weak Cryptography, Insufficient Logging, Missing

- **low**: 보안 Best Practice 위반, 간접적 위험
  예: Code Quality Issues, Minor Configuration Issues
```

How

```python
def analyze_file_dependencies(file_contents: Mapping[str, str]) -> Dict[str, Set[str]]:
    """파일 간의 의존성을 분석하여 연관 관계를 파악합니다."""
    dependencies = defaultdict[Any, set](set)

    # 각 언어별 import/include 패턴
    import_patterns = {
        # Java
        'java': [
            r'import\s+([a-zA-Z_][a-zA-Z0-9_.]*);',
            r'import\s+static\s+([a-zA-Z_][a-zA-Z0-9_.]*);',
        ],
        # Python
        'py': [
            r'from\s+([a-zA-Z_][a-zA-Z0-9_.]*)\s+import',
            r'import\s+([a-zA-Z_][a-zA-Z0-9_.]*)',
        ],
        # JavaScript/TypeScript
        'js': [
            r'import.*from\s+[\'"]([^\'"]*)[\'"]\s*;?',
            r'require\s*\(\s*[\'"]([^\'"]*)[\'"]\s*\)',
            r'import\s*\(\s*[\'"]([^\'"]*)[\'"]\s*\)',
        ],
        'ts': [
            r'import.*from\s+[\'"]([^\'"]*)[\'"]\s*;?',
            r'require\s*\(\s*[\'"]([^\'"]*)[\'"]\s*\)',
            r'import\s*\(\s*[\'"]([^\'"]*)[\'"]\s*\)',
        ],
```

```python
def extract_file_structure(content: str, language: str) -> str:
    """파일의 구조적 정보 추출 (클래스, 함수 등)"""
    lines = content.split('\n')
    structure = []

    patterns = {
        # 주요 지원 언어
        'python': [
            (r'^\s*class\s+(\w+)', 'Class'),
            (r'^\s*def\s+(\w+)', 'Function'),
            (r'^\s*async\s+def\s+(\w+)', 'Async Function'),
            (r'^\s*@\w+', 'Decorator')
        ],
        'java': [
            (r'^\s*(?:public\s+|private\s+|protected\s+)?(?:static\s+)?class\s+(\w+)', 'Class'),
            (r'^\s*(?:public\s+|private\s+|protected\s+)?interface\s+(\w+)', 'Interface'),
            (r'^\s*(?:public\s+|private\s+|protected\s+)?(?:static\s+)?enum\s+(\w+)', 'Enum'),
            (r'^\s*(?:public\s+|private\s+|protected\s+)?(?:static\s+)?\w+\s+(\w+)\s*\(', 'Method'),
            (r'^\s*@\w+', 'Annotation')
        ],
        'javascript': [
            (r'^\s*class\s+(\w+)', 'Class'),
            (r'^\s*function\s+(\w+)', 'Function'),
            (r'^\s*const\s+(\w+)\s*=.*=>', 'Arrow Function'),
            (r'^\s*export\s+(?:default\s+)?(?:class|function)\s+(\w+)', 'Export'),
            (r'^\s*async\s+function\s+(\w+)', 'Async Function')
        ],
        'typescript': [
            (r'^\s*class\s+(\w+)', 'Class'),
            (r'^\s*interface\s+(\w+)', 'Interface'),
            (r'^\s*type\s+(\w+)', 'Type'),
            (r'^\s*enum\s+(\w+)', 'Enum'),
            (r'^\s*function\s+(\w+)', 'Function'),
            (r'^\s*const\s+(\w+)\s*=.*=>', 'Arrow Function'),
            (r'^\s*export\s+(?:default\s+)?(?:class|interface|function)\s+(\w+)', 'Export')
        ],
        'kotlin': [
            (r'^\s*class\s+(\w+)', 'Class'),
            (r'^\s*data\s+class\s+(\w+)', 'Data Class'),
            (r'^\s*sealed\s+class\s+(\w+)', 'Sealed Class'),
            (r'^\s*interface\s+(\w+)', 'Interface'),
            (r'^\s*enum\s+class\s+(\w+)', 'Enum'),
            (r'^\s*object\s+(\w+)', 'Object'),
            (r'^\s*fun\s+(\w+)', 'Function'),
            (r'^\s*suspend\s+fun\s+(\w+)', 'Suspend Function'),
            (r'^\s*@\w+', 'Annotation')
        ],
```

Result

🔒 보안 취약점 상세 분석

📊 검사 결과 요약

분류	Semgrep	AI 추가 발견	총계
🚨 치명적/높음	4개	2개	**6개**
🟡 보통	0개	3개	3개
ℹ️ 낮음	0개	0개	0개
✅ 오탐	0개	-	0개

📈 위험도 분포:

- 🚨 즉시 조치 필요: 6개
- 🟡 검토 권장: 3개
- ℹ️ 참고사항: 0개

스캔 완료: 2025-10-28 09:50:39 UTC

🚨 즉시 수정 필요 - Semgrep 발견 (4

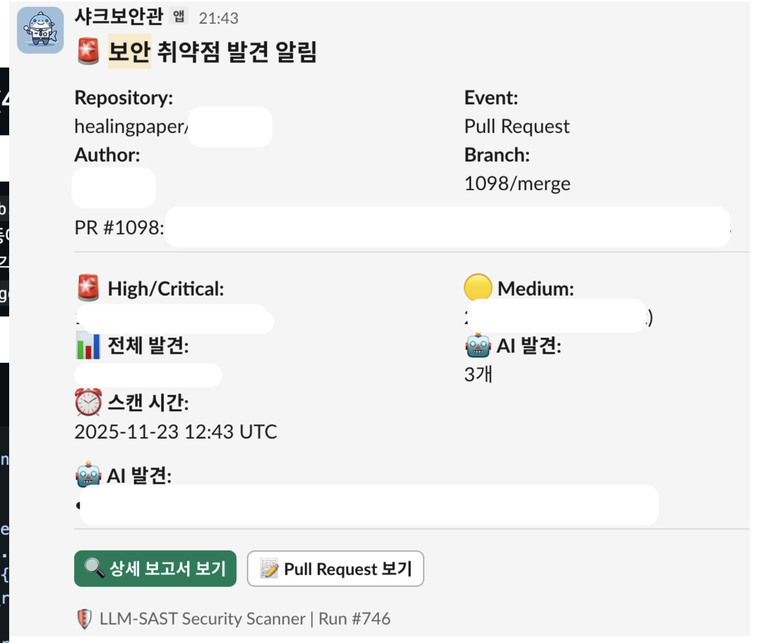
1. 🚨 ERROR -

🔍 문제점: GitHub Actions에서 `${{ github`
다. 공격자가 PR 제목, 브랜치명, 커밋 메시지 등
취, 코드 변조, 인프라 침해로 이어질 수 있는 심각
`steps.changed_files.outputs.all_chang`

📍 위치:

🔍 코드 컨텍스트:

```
73:         - name: Determine scan
74:           id: targets
75: →         run: |
76: →           if [ "${{ github.e
77: →             if [ "${{ steps.
78: →               echo "files=${
79: →               echo "should_r
80: →             else
81: →               echo "should_run=false" >> $GITHUB_OUTPUT
82: →             fi
83: →           else
84: →             echo "files=${{ inputs.files }}" >> $GITHUB_OUTPUT
85: →             echo "should_run=true" >> $GITHUB_OUTPUT
86: →           fi
87: →
88:         - name: Setup scan environment
89:           if: steps.targets.outputs.should_run == 'true'
```



이게 은근 도움 많이 되더라구요. 바트 고생 많으십니다!

1 감사합니다 1

54 minutes ago
2m 54s

1 hour ago
2m 23s

Today at 7:55 PM
2m 52s

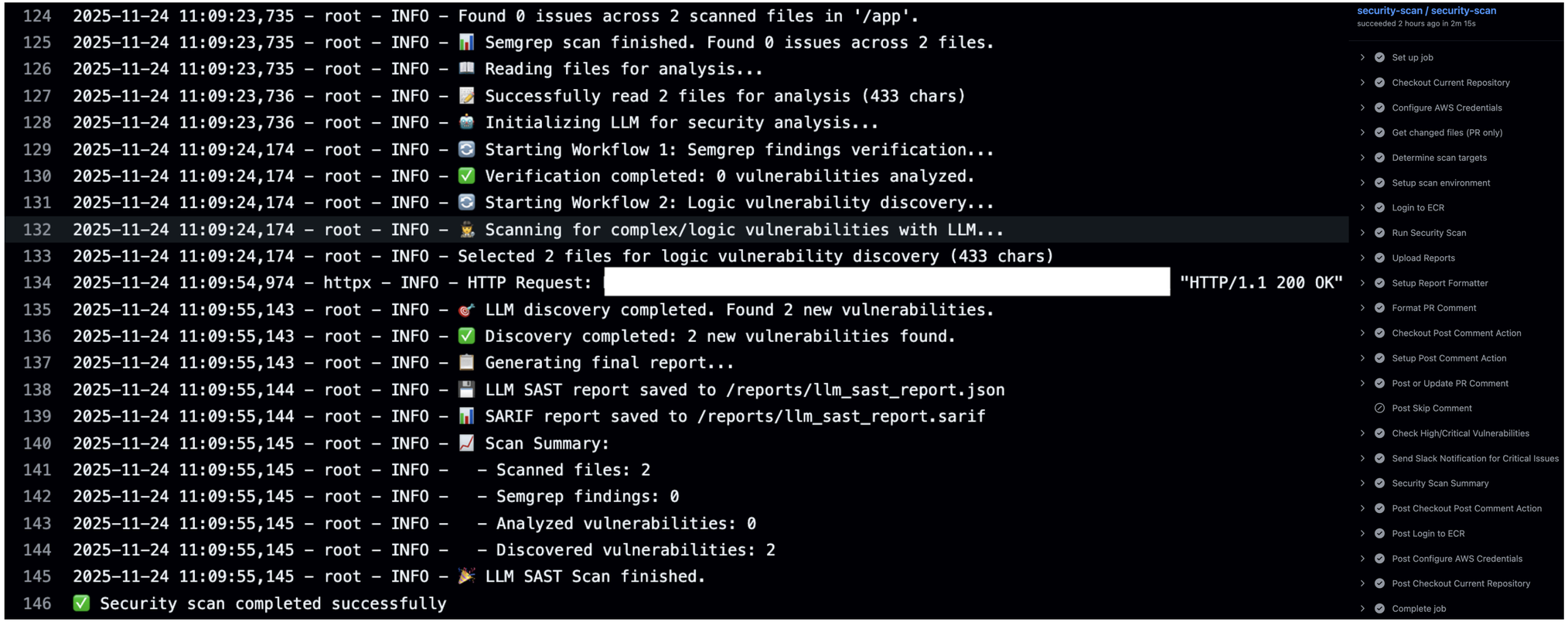
Today at 7:53 PM
2m 39s

Today at 7:52 PM
2m 28s

Today at 2:58 PM
2m 51s

Today at 1:36 PM
3m 10s

Result



```
124  2025-11-24 11:09:23,735 - root - INFO - Found 0 issues across 2 scanned files in '/app'.
125  2025-11-24 11:09:23,735 - root - INFO - 📊 Semgrep scan finished. Found 0 issues across 2 files.
126  2025-11-24 11:09:23,735 - root - INFO - 📖 Reading files for analysis...
127  2025-11-24 11:09:23,736 - root - INFO - 📝 Successfully read 2 files for analysis (433 chars)
128  2025-11-24 11:09:23,736 - root - INFO - 🤖 Initializing LLM for security analysis...
129  2025-11-24 11:09:24,174 - root - INFO - 🔄 Starting Workflow 1: Semgrep findings verification...
130  2025-11-24 11:09:24,174 - root - INFO - ✅ Verification completed: 0 vulnerabilities analyzed.
131  2025-11-24 11:09:24,174 - root - INFO - 🔄 Starting Workflow 2: Logic vulnerability discovery...
132  2025-11-24 11:09:24,174 - root - INFO - 🥷 Scanning for complex/logic vulnerabilities with LLM...
133  2025-11-24 11:09:24,174 - root - INFO - Selected 2 files for logic vulnerability discovery (433 chars)
134  2025-11-24 11:09:54,974 - httpx - INFO - HTTP Request: [redacted] "HTTP/1.1 200 OK"
135  2025-11-24 11:09:55,143 - root - INFO - 🎯 LLM discovery completed. Found 2 new vulnerabilities.
136  2025-11-24 11:09:55,143 - root - INFO - ✅ Discovery completed: 2 new vulnerabilities found.
137  2025-11-24 11:09:55,143 - root - INFO - 📋 Generating final report...
138  2025-11-24 11:09:55,144 - root - INFO - 💾 LLM SAST report saved to /reports/llm_sast_report.json
139  2025-11-24 11:09:55,144 - root - INFO - 📊 SARIF report saved to /reports/llm_sast_report.sarif
140  2025-11-24 11:09:55,145 - root - INFO - 📈 Scan Summary:
141  2025-11-24 11:09:55,145 - root - INFO -   - Scanned files: 2
142  2025-11-24 11:09:55,145 - root - INFO -   - Semgrep findings: 0
143  2025-11-24 11:09:55,145 - root - INFO -   - Analyzed vulnerabilities: 0
144  2025-11-24 11:09:55,145 - root - INFO -   - Discovered vulnerabilities: 2
145  2025-11-24 11:09:55,145 - root - INFO - 🎉 LLM SAST Scan finished.
146  ✅ Security scan completed successfully
```

Result

5분 → 2.5분
유지보수성 UP
취약점 퀄리티 UP
오탐율 DOWN
비용 DOWN 1/3
보안 코드 리뷰 활성화

Tip

워크플로우

- Github Action 워크플로우 동작 시 bot 예외처리 (예: renovate)
- SAST 메인 레포지토리 만들고 타 레포지토리에서 재사용할 수 있는 워크플로우 형태로 배포

```
jobs:
  security-scan:
    uses: code-security-repo/.github/actions/code-security-reusable.yaml@main
```

에이전트

- 에이전트별로 결과 로그 남기고 디버깅/트레이싱 할 수 있게 구성

Knowledge

- 프로젝트에 아키텍처 설명 작성 readme.md, architecture_explaination.md 등 이를 프롬프트에 활용

본인이 관리하는 프로젝트 규모 파악 및 LLM 기반 자동화 시 소스코드에 대한 보안 리스크(코드 내 Secret 등) 리뷰 필요

Next Features

스캔 결과 DB화 및 대시보드

개발자 피드백 루프 (Human-in-the-loop)

정규식 파싱 → AST 파싱 전환 (tree-sitter 활용 예정)

변경 파일 의존성 세부 확인 및 참고 로직 개선

자동 Jira 티켓 생성 및 완료 처리 기능

사내 데이터(*.md) 등 Knowledge base 강화 및 취약점 결과 기반 RAG 구성 (중요도 및 퀄리티 향상, 피드백 루프 활용)

“Aardvark, OpenAI의 에이전틱 보안 리서처” 따라해보기

Key takeways

1. LLM으로 SAST의 근본적인 한계(맥락 부족)를 극복하기
2. Simple is Best: 복잡한 멀티 에이전트 대신 One-Shot 전략을 고려해보기
3. 개발자 경험을 해치지 않는 방식으로 보안을 통합해보기
4. 전략적 구축: 확신을 얻기위해 Small Start 타협점 확인하기
5. 시장 트렌드 파악하고 알맞는 솔루션 적용해보기

Q&A