AI기반 취약점 탐지 워크플로우 구축기

Building a Multi-LLM Vulnerability Detection Workflow

신현서

OWASP Seoul · 2026 May

Korea University

WHO AM I

신현서

고려대학교 / Korea University

CYKOR / 고려대 정보보호 동아리

BOB 14th / 차세대 보안리더 양성 프로그램

Interested in **web hacking** and **AI for security**

AGENDA

목차

WHY & RESULTS

AI로 취약점을 찾게 된 계기 및 성과

01

WHY THIS PROJECT

이 워크플로우를 만든 이유

How AI for security has evolved over the past year

2024.11

MCP
표준 프로토콜

Anthropic이 공개한 AI 도구 연결 표준

2025.06

XBOW
HackerOne US 1위

자율 AI가 90일 만에 1,060개 취약점 보고

2025.08

AIxCC
한국팀 우승

Team Atlanta(KAIST 외) $4M 상금

2025 - 2026

CLI Tools
에이전트 코딩

Claude Code, Cursor, Codex 등 보편화

AI가 보안을 바꾸기 시작했다

WHY LISTEN

이 워크플로우로 거둔 성과

Real 0-days in production open-source

01 / CVES ASSIGNED

17

CVE 발급

Grafana, Airflow, Discourse, Nextcloud, protobuf, n8n

02 / BUG BOUNTIES

49

버그바운티 보상

HackerOne, Intigriti, YesWeHack, Bugcrowd, Huntr

03 / HIGHEST CVSS

8.1

최고 심각도

Privilege Escalation in Grafana

04 / HACKERONE KOREA

#1

HackerOne 1위

Korea Reputation #1

02

MAS VS WORKFLOW

멀티 에이전트 vs 워크플로우

APPROACH

워크플로우 vs 멀티 에이전트

Two approaches to LLM-based vulnerability detection

WORKFLOW

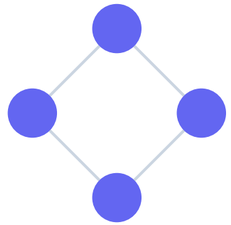
개발자가 흐름을 결정

개발자가 흐름을 미리 설계한다

The developer defines every step in advance.

MULTI-AGENT

LLM들이 스스로 결정



LLM들이 스스로 결정한다

Agents observe, reason, and decide on their own.

APPROACH

워크플로우 vs 멀티 에이전트

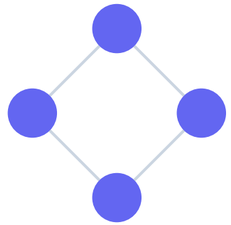
Two approaches to LLM-based vulnerability detection

WORKFLOW

개발자가 흐름을 결정

MULTI-AGENT

LLM들이 스스로 결정

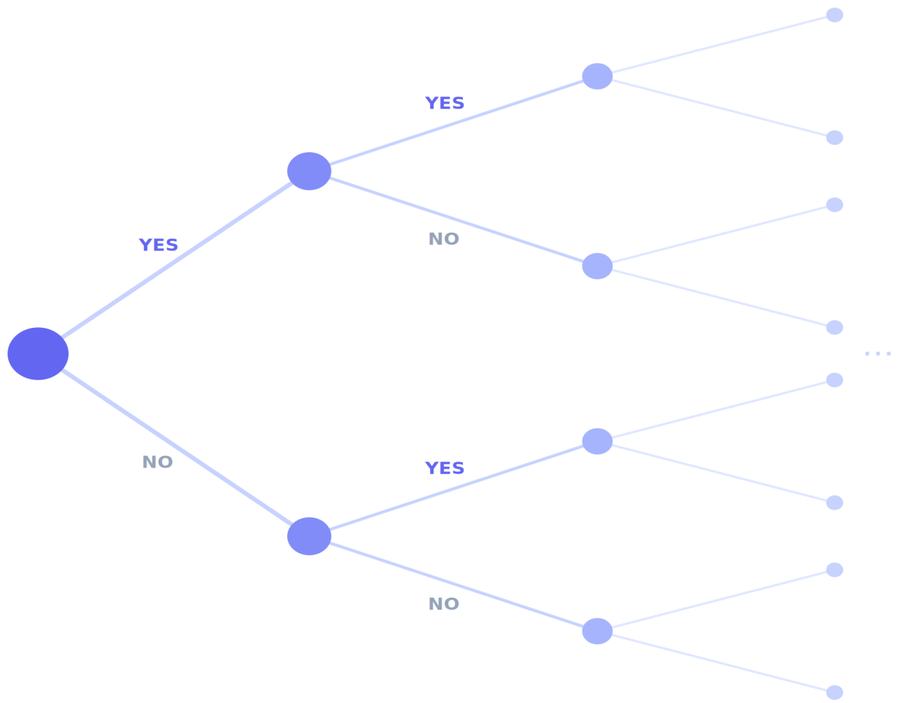


CRITERIA	WORKFLOW	MULTI-AGENT
누가 결정하나	개발자가 사전에 설계	LLM들이 동적으로 판단
비용	저렴함 (호출 횟수 제한)	고비용 (호출 횟수 폭증)
디버깅	쉬움 (예측 가능한 흐름)	어려움 (동작 예측 불가)
복잡한 체이닝	어려움	가능

WORKFLOW PROBLEM

사전 정의의 한계

블랙박스 환경에서 워크플로우를 직접 설계하면 생기는 문제




```
needs_login(url)
```

접속해보기 전까지는 알 수 없다

```
waf_active(resp)
```

페이로드를 시도하기 전까지는 알 수 없다

```
find_inputs(page)
```

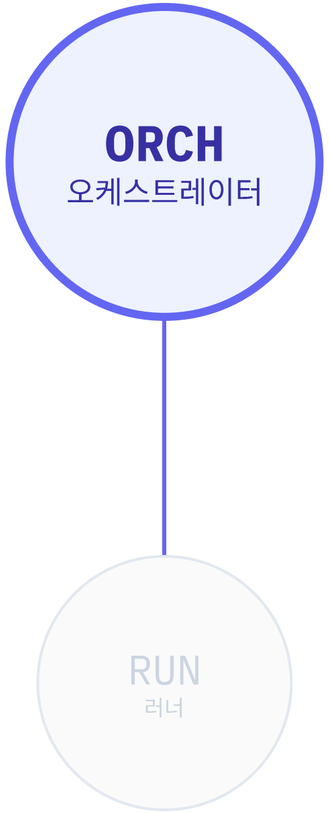
탐색하기 전까지는 알 수 없다

```
csrf_needed(form)   # ???
rate_limited(ip)    # ???
session_scope(usr)  # ???
```

무엇을 해야 하는지는 응답을 보고 나서야 알 수 있다.
그래서 워크플로우로는 불가능하다.

AGENT FLOW / CALL 01

서비스 분석 — 첫 번째 태스크 설정



ACTIVE: ORCH

INPUT

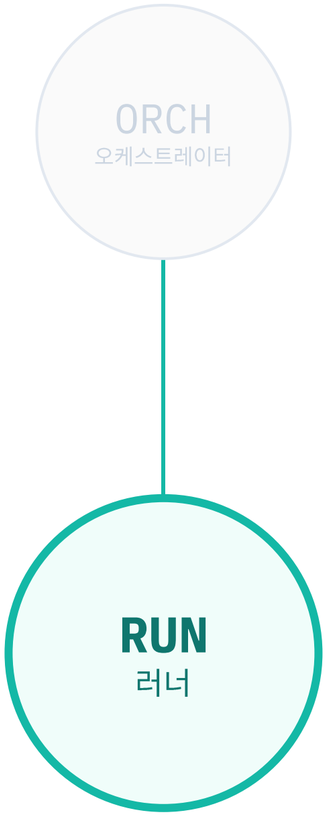
서비스 URL 하나만 주어진 상태
무엇부터 시작할지 결정해야 한다

DECISION

로그인이 필요한 서비스인지 확인한다

AGENT FLOW / CALL 02

Runner 호출 — 회원가입 및 계정 확보



ACTIVE: RUN

PROMPT

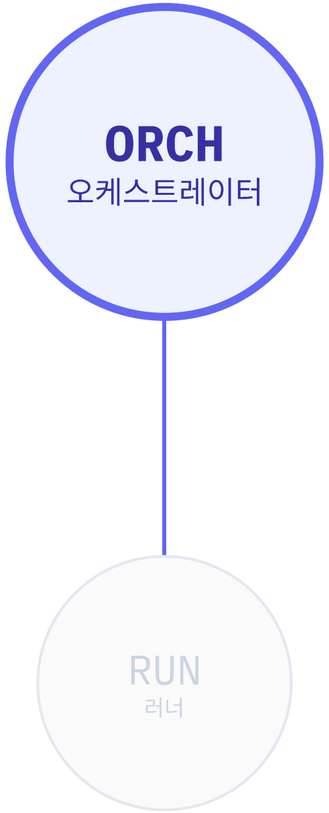
로그인이 필요한 서비스인지 확인해줘

RESULT

로그인하지 않으면 글을 쓸 수 없는 서비스

AGENT FLOW / CALL 03

Orchestrator 결정 — 회원가입 위임



ACTIVE: ORCH

INPUT

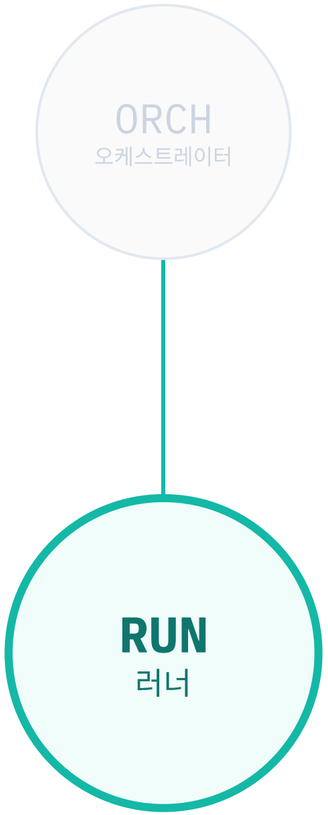
로그인하지 않으면 글을 쓸 수 없는 서비스

DECISION

회원가입부터 시작한다

AGENT FLOW / CALL 04

Runner 호출 — 회원가입 및 계정 확보



ACTIVE: RUN

PROMPT

회원가입하고

id / pw 를 확보해줘

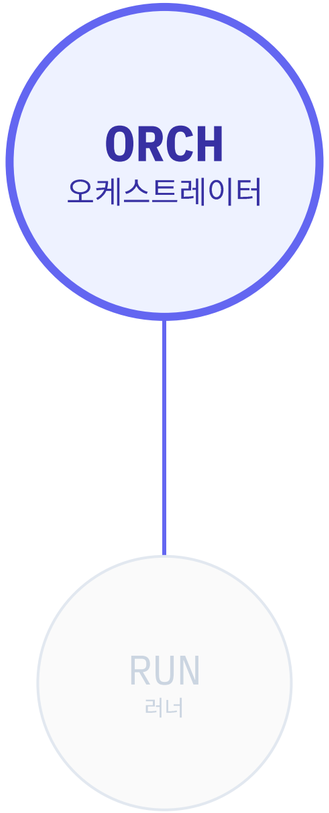
RESULT

계정 생성 완료

test@test.com / test1234

AGENT FLOW / CALL 05

Orchestrator — 다음 태스크 결정



ACTIVE: ORCH

OBSERVED

계정 생성 완료

test@test.com / test1234 확보

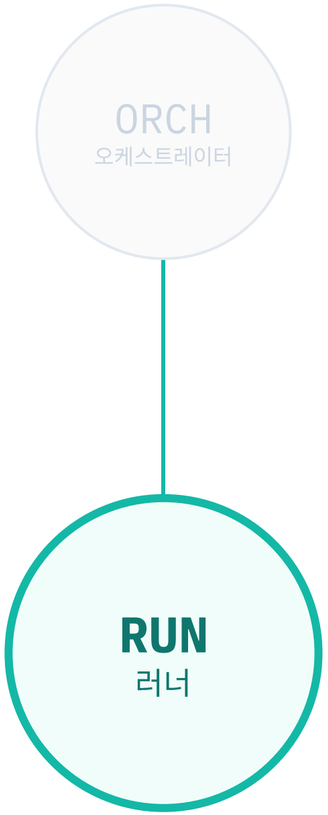
DECISION

로그인 후 입력 가능한 화면을 파악해야 한다

→ Runner 에게 입력면 탐색 위임

AGENT FLOW / CALL 06

Runner 호출 — 로그인 후 입력면 탐색



ACTIVE: RUN

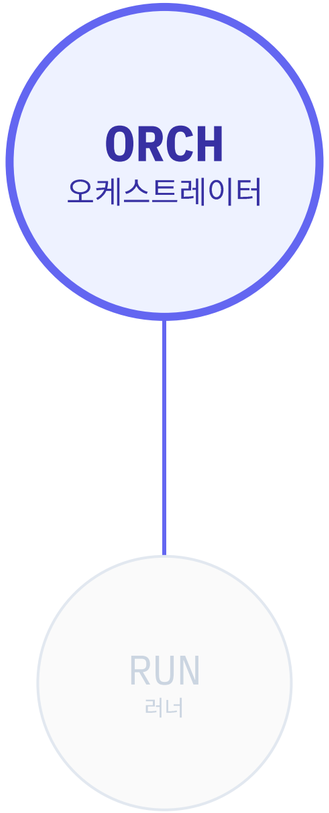
PROMPT

로그인하고
XSS 를 시도해볼만한 입력면을 찾아줘

RESULT

/profile, /post/new, /comment
XSS 가능성 있는 입력면 발견

Orchestrator 결정 — WAF 조사 위임



ACTIVE: ORCH

OBSERVED

입력 지점 3개 확보

WAF 존재 여부와 렌더링 컨텍스트를 아직 모른다

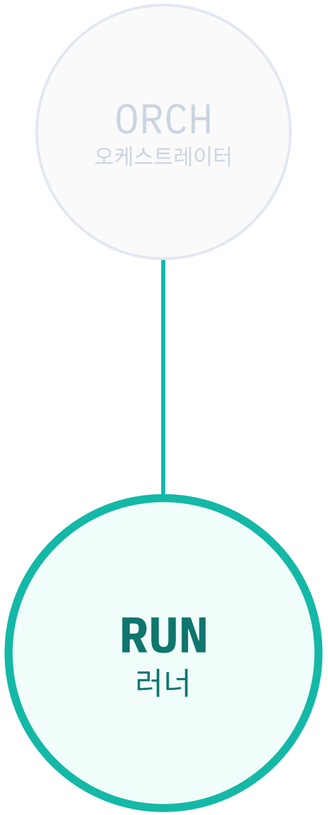
DECISION

WAF 나 이스케이프 여부를 알 수 없다

→ Runner 에게 조사 위임

AGENT FLOW / CALL 08

Runner 실행 — WAF 및 컨텍스트 조사



ACTIVE: RUN

PROMPT

WAF 반응과 렌더링 컨텍스트를 조사해줘

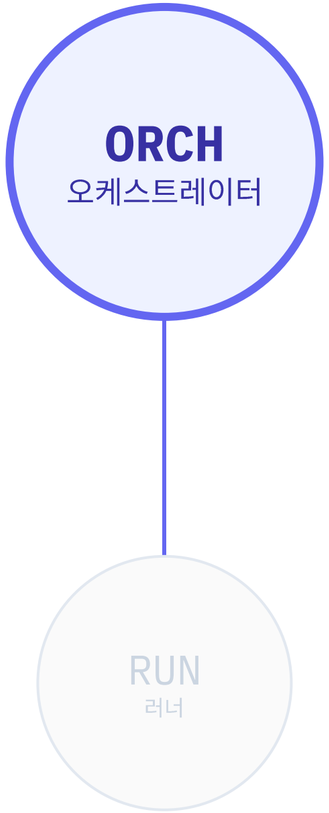
RESULT

/profile, /post/new → 이스케이프 처리, 원천 차단

/comment → WAF 존재, 이스케이프 없음

AGENT FLOW / CALL 09

Orchestrator 결정 — 검증 대상 선별



ACTIVE: ORCH

OBSERVED

/profile, /post/new → 이스케이프로 원천 차단

/comment → WAF 존재, 이스케이프 없음

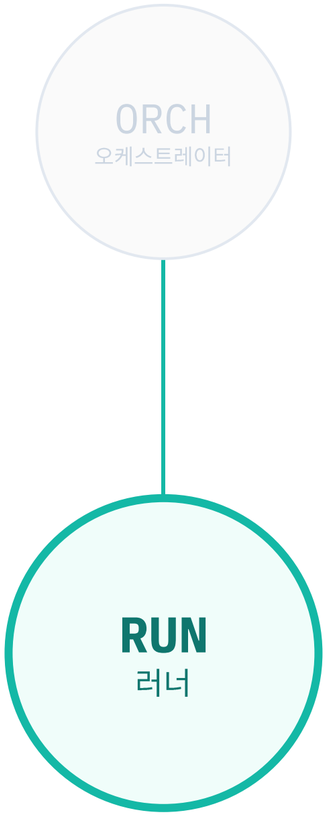
DECISION

/comment 가 유일한 대상

WAF 우회 페이로드로 XSS 시도 필요

AGENT FLOW / CALL 10

Runner 실행 — XSS 페이로드 검증



ACTIVE: RUN

PROMPT

/comment 에 WAF 우회 XSS 페이로드를 시도해줘

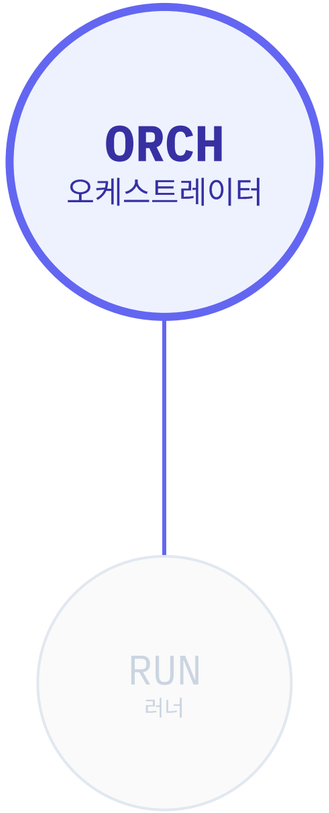
RESULT

WAF 우회 성공 → alert(1) 실행 확인

Stored XSS 취약점 확인

AGENT FLOW / CALL 11

Orchestrator 결정 — 임팩트 파악 시작



ACTIVE: ORCH

OBSERVED

Stored XSS 확인

실제 공격 가능 여부를 파악해야 한다

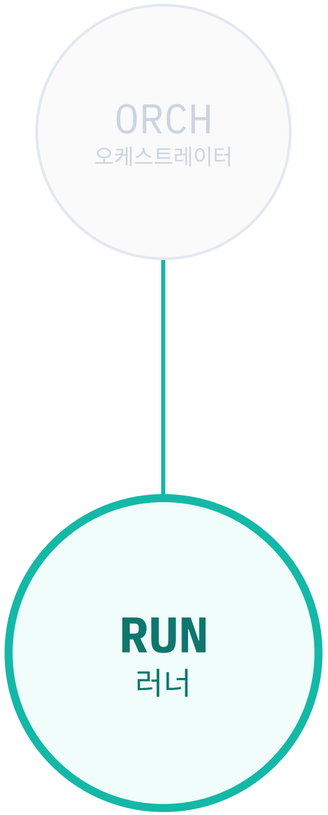
DECISION

쿠키 탈취 가능 여부부터 확인

→ Runner 에게 HttpOnly 플래그 확인 위임

AGENT FLOW / CALL 12

Runner 실행 — 쿠키 탈취 가능 여부 확인



ACTIVE: RUN

PROMPT

세션 쿠키에 HttpOnly 플래그가 있는지 확인해줘

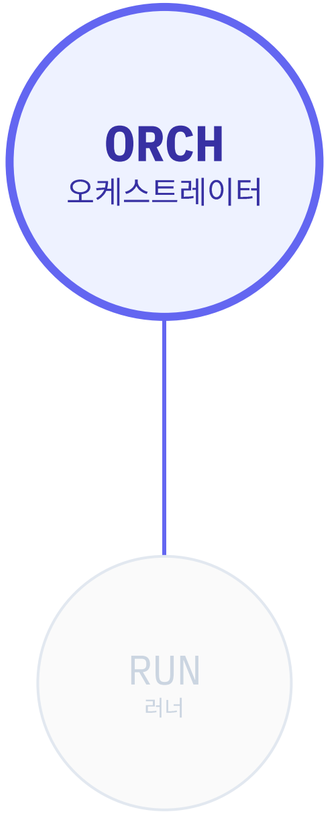
RESULT

HttpOnly 플래그 존재

쿠키 탈취 불가

AGENT FLOW / CALL 13

Orchestrator 결정 — 대안 임팩트 탐색



ACTIVE: ORCH

OBSERVED

쿠키 탈취 불가

다른 경로로 계정 탈취가 가능한지 확인 필요

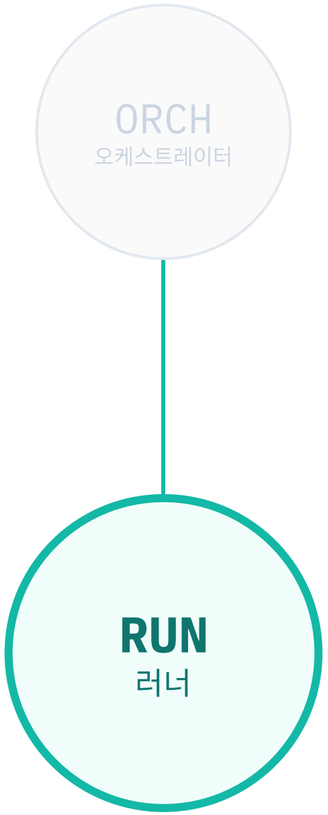
DECISION

XSS로 CSRF가 가능한 엔드포인트 중

의미 있는 것을 찾아봐 → Runner에게 위임

AGENT FLOW / CALL 14

Runner 실행 — 패스워드 변경 CSRF 시도



ACTIVE: RUN

PROMPT

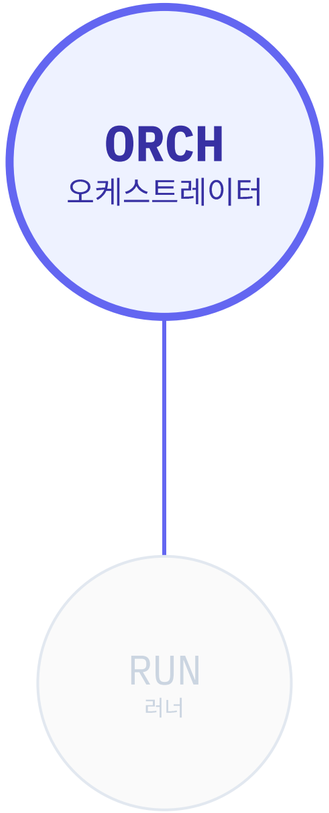
XSS를 통해 CSRF 가능한 엔드포인트 중
실제 피해로 이어질 수 있는 것을 찾아줘

RESULT

패스워드 변경 엔드포인트 → CSRF 토큰 없음
공격 시 계정 탈취로 이어질 수 있음

AGENT FLOW / CALL 15

Orchestrator 결정 — 패스워드 변경 CSRF 시도



ACTIVE: ORCH

OBSERVED

패스워드 변경 엔드포인트 → CSRF 토큰 없음

계정 탈취로 이어질 수 있음

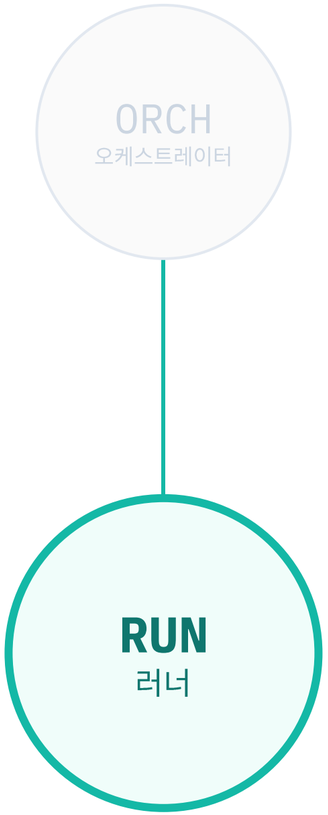
DECISION

패스워드 변경 CSRF 를 실제로 시도

→ Runner 에게 공격 실행 위임

AGENT FLOW / CALL 16

Runner 실행 — CSRF 공격 및 계정 탈취 확인



ACTIVE: RUN

PROMPT

XSS 를 통해 패스워드 변경 CSRF 를 실행해줘

RESULT

CSRF 성공 → 패스워드 변경 완료

XSS + CSRF = 계정 탈취 확인

WORKFLOW EXAMPLE

실제로 만든 워크플로우

웹 오픈소스 코드 분석에 사용한 공통 3단계

03

TOOL SELECTION

왜 워크플로우였나

BLACKBOX 서비스 펜테스팅

URL만 주어진 상태

응답을 보고 나서야
다음 행동을 결정할 수 있다

경로를 탐색하고 상태를 추적하며
복잡한 체이닝이 필요하다

WHITEBOX 웹 오픈소스 코드 분석

코드가 주어진 상태

처음부터 모든 정보가
주어진다

탐색이 필요 없고
공격의 시작점이 되는 취약점만 찾으면 된다
패턴 탐지로 충분하다

웹 오픈소스 코드 분석은 워크플로우로 충분했다

COMMON STRUCTURE

어떤 워크플로우든 결국 같은 3단계

The 3 steps every workflow shared

STEP 01

분석 대상 전처리

Preprocessing

STEP 02

취약점 가설 생성

Hypothesis Generation

STEP 03

취약점 검증

Verification

STAGE 00 / 03

분석할 레포지토리 준비

Initial State — Target Repository

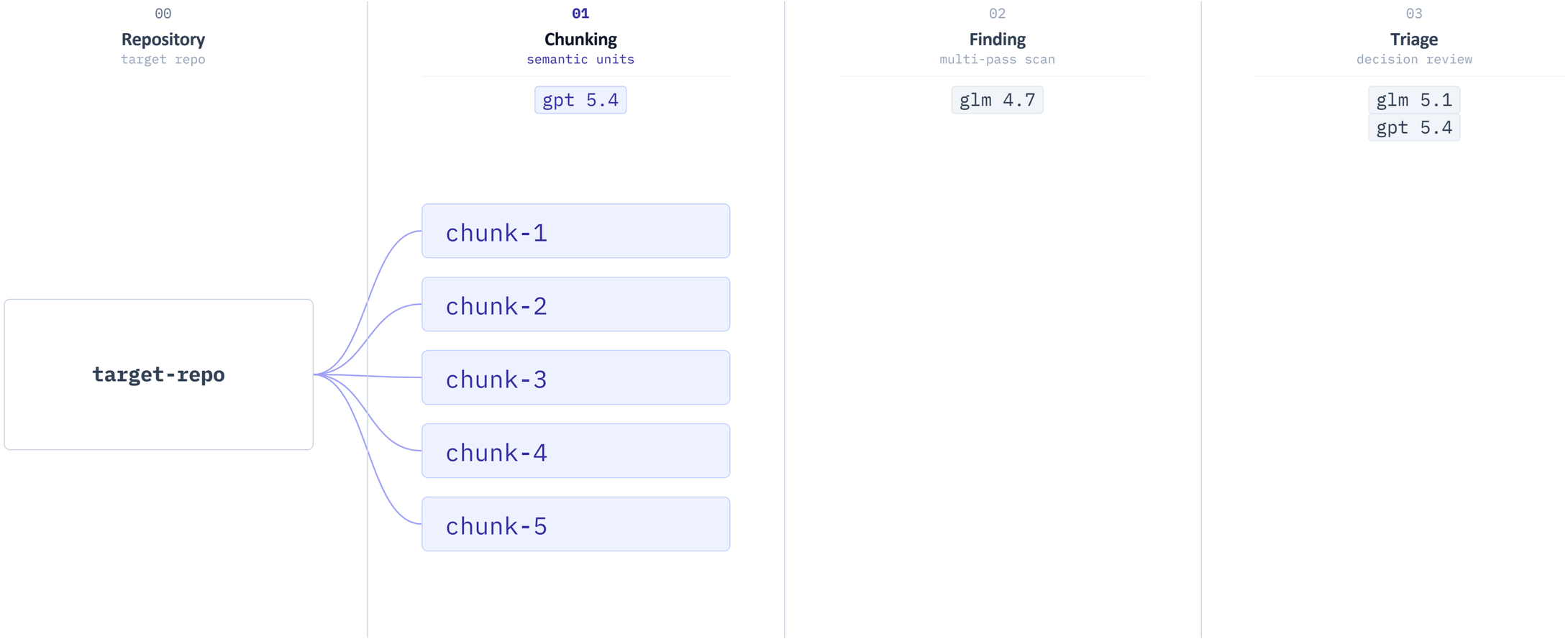
00	01	02	03
Repository	**Chunking**	**Finding**	**Triage**
target repo	semantic units	multi-pass scan	decision review
	`gpt 5.4`	`glm 4.7`	`glm 5.1` `gpt 5.4`

분석 대상 레포지토리가 주어진 **시작 상태**. 이후 단계의 입력으로 사용될 코드베이스를 확정한다.

STAGE 01 / 03

분석 단위로 코드 분할

Semantic Chunking

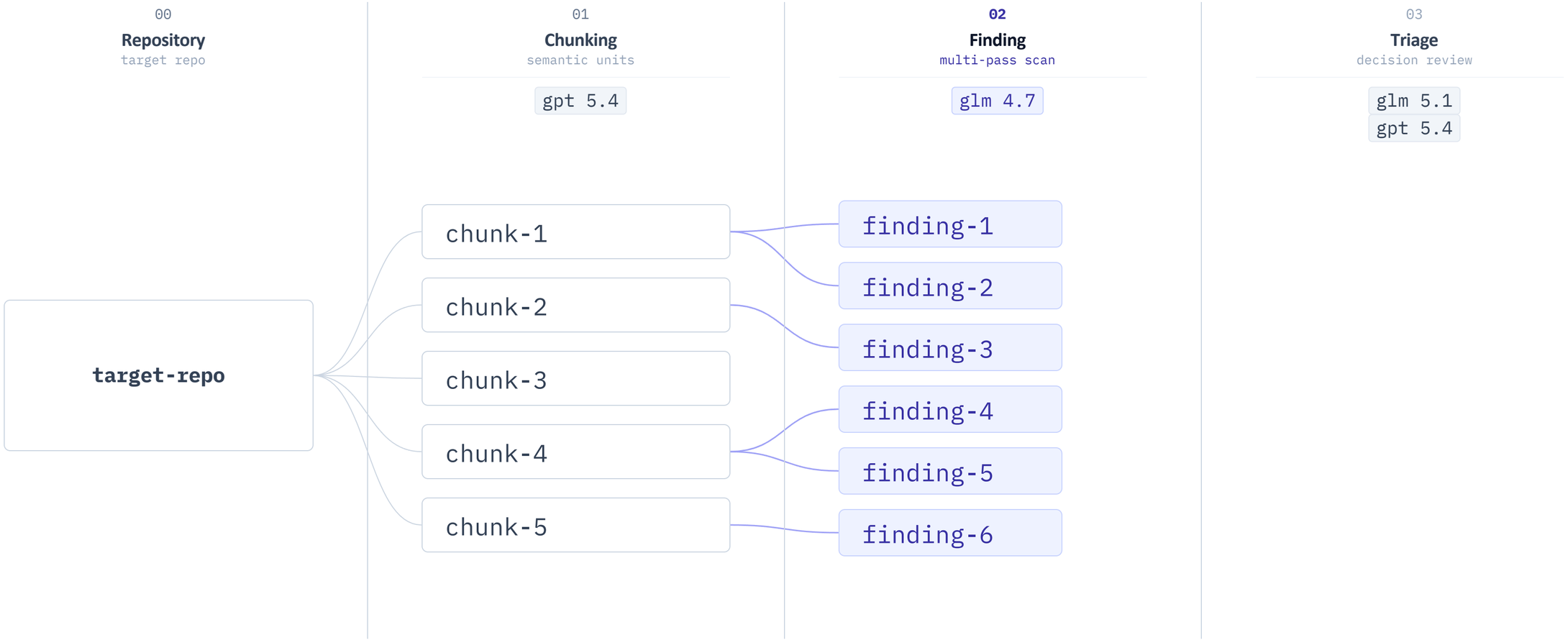


코드베이스를 **LLM 컨텍스트에 적합한 의미 단위**로 분해해 병렬 분석 가능한 형태로 정규화한다.

STAGE 02 / 03

취약점 후보 탐지

Multi-pass Vulnerability Finding

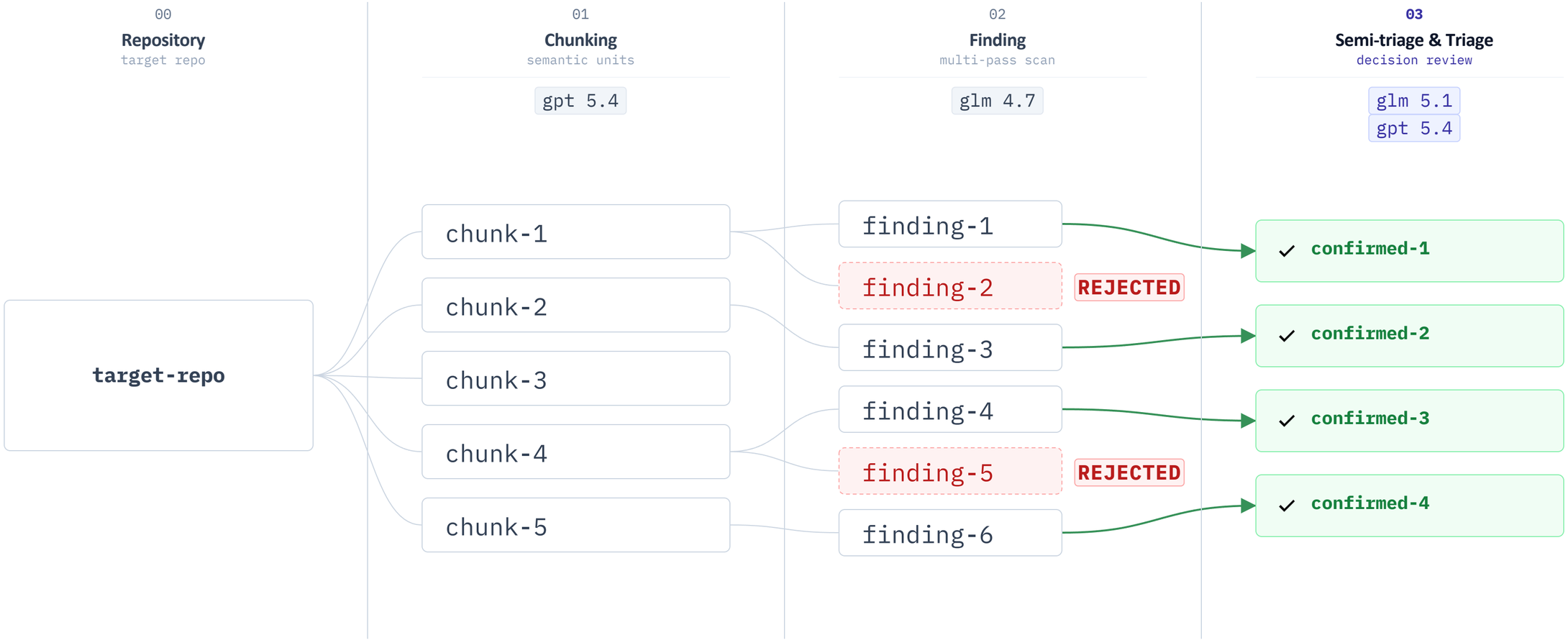


각 청크에 대해 **LLM 분석**을 수행하여 잠재 취약점 후보를 광범위하게 추출한다.

STAGE 03 / 03

이중 검증을 통한 최종 확정

Triage Pipeline (Semi-triage + Final Review)



Semi-triage가 False Positive를 1차 제거하고, **Triage**가 후보를 재검토하여 실제 취약점 여부를 확정한다.

LIMITATIONS

AI 취약점 탐지의 한계

04

LIMITATIONS

AI 취약점 탐지의 한계

01

버그헌팅 관점에서는 아직 비싸다

구독제가 없었다면 개인 수준에선 운용 불가

02

취약점의 그레이존

코드가 아니라 정책이 취약점 여부를 결정한다

03

취약점의 책임 경계

책임이 여러 컴포넌트 사이로 분산된 취약점

LIMITATION 01 / COST

버그헌팅 관점에서는 아직 비싸다

Why individual bug hunters cannot sustain this workflow

청크 생성

레포당 수백 개의 청크가 나온다

중간 규모 오픈소스 레포 하나에 의미 단위 청크 수백 개 발생

호출 증가

한 레포당 수백~수천 번의 LLM 호출

청크 수 × 청크당 여러 호출

ROI 계산

개인 버그헌터에겐 적자 구조

보상은 대부분 0 · 운 좋으면 $1,000 정도 · 비용은 레포당 수백~수천 달러

LIMITATION 02 / GRAY ZONE

취약점의 그레이존

LLM lacks intuition to recognize policy-dependent decisions

패턴 감지

LLM은 코드 패턴으로만 위험도를 판단한다

SSRF, RCE, IDOR 같은 패턴을 보면 정책 맥락 없이 바로 취약점으로 검출

정책 공백

취약점 여부는 정책이 결정한다

같은 코드도 프로젝트 정책에 따라 의도된 기능일 수도, 버그일 수도 있다

직감 부재

사람은 직감하거나 회의를 하지만 LLM은 그럴 수 없다

사람이라면 팀에 묻거나 회의로 정할 일을, LLM은 그대로 취약점으로 분류한다

LIMITATION 02 / GRAY ZONE — CASE

예시 서비스

사용자별 AI 토큰 한도가 있는 SaaS

사용자는 매월 정해진 토큰량을 부여받고, 한도 초과 시 새 요청이 차단되는 구조

비즈니스 룰

매월 사용자별 토큰량 충전

한도 초과 시 새 요청은 차단

진행 중인 요청은 완료를 보장

사용 사례

AI 챗봇 서비스

AI 글쓰기·요약 도구

AI 코드 어시스턴트, 이미지 생성

LIMITATION 02 / GRAY ZONE — ANALYSIS

사람과 LLM은 다르게 본다

발생한 일

한도의 99%를 소진한 사용자가 새 요청을 보냈고, 서비스는 한도를 넘은 뒤에도 끝까지 처리했다
사용자에게 주어진 토큰 제한을 넘어서 토큰을 소비했다

HUMAN

사람은 정책을 먼저 찾는다

01 정책 문서를 확인한다

02 정책에 있다 → 의도된 동작

03 정책에 없다 → 회의·논의로 판단

LLM

LLM은 코드만 본다

01 정책 문서를 읽을 수 없다

02 코드만 보고 판단한다

03 한도 초과 동작 → 무조건 취약점

LIMITATION 03 / RESPONSIBILITY BOUNDARY

취약점의 책임 경계

LLM sees one repo — responsibility across services is invisible

레포 단위

한 레포를 볼 때, 연결된 다른 레포는 보지 못한다

실제 서비스는 여러 SaaS·서버가 연결돼 동작하지만, 분석 중인 레포 밖의 코드는 보지 못한다. 설사 볼 수 있더라도 어느 단에서 막아야 하는지가 애매하다.

책임 경계

어느 서비스에서 막아야 하는지 알 수 없다

이스케이프 같은 보안 처리 책임이 어느 컴포넌트에 있는지는 코드만으로 판단할 수 없다

판단 불가

LLM은 "이건 다른 서비스 책임"이라는 결론을 잘 내리지 못한다

한 레포 밖을 볼 수 없어 컴포넌트 간 계약에 기반한 결론을 내리지 못하고, 양쪽 레포에 취약점으로 검출한다

LIMITATION 03 / RESPONSIBILITY BOUNDARY — CASE

예시 서비스

메일 발송 기능이 있는 SaaS

SaaS 서비스가 메일 서버를 통해 수신자에게 알림 메일을 보내는 구조

구성 요소

SaaS 서비스 — 메일 내용 생성·발송 요청

메일 서버 — 수신자에게 메일 중계

메일 클라이언트 — HTML 렌더링 후 표시

사용 사례

알림 메일, 비밀번호 재설정 메일

마케팅·뉴스레터 발송 도구

이슈 트래커, 협업 툴 알림

LIMITATION 03 / RESPONSIBILITY BOUNDARY — CASE

누구에게 이스케이프 책임이 있는가?

HUMAN

취약점이 아닐 수 있다

01 메일 서버도 함께 살펴본다

02 메일 서버가 이스케이프할 것으로 기대

03 명세·계약 확인 후 판단한다

LLM

취약점으로 검출한다

01 메일 서버·클라이언트를 분석하지 못한다

02 SaaS 코드만 보고 "이스케이프 없음"

03 설사 분석해도 책임이 어디에 있는지 애매하다

Thank you

CONTACT · 신현서 · se1en

LINKEDIN linkedin.com/in/hyunseo-shin-bb074632a

BLOG se1en.tistory.com/4

EMAIL selen0328@gmail.com