취약점을 판정하는 모델에서 가설을 반증하는 Agent로

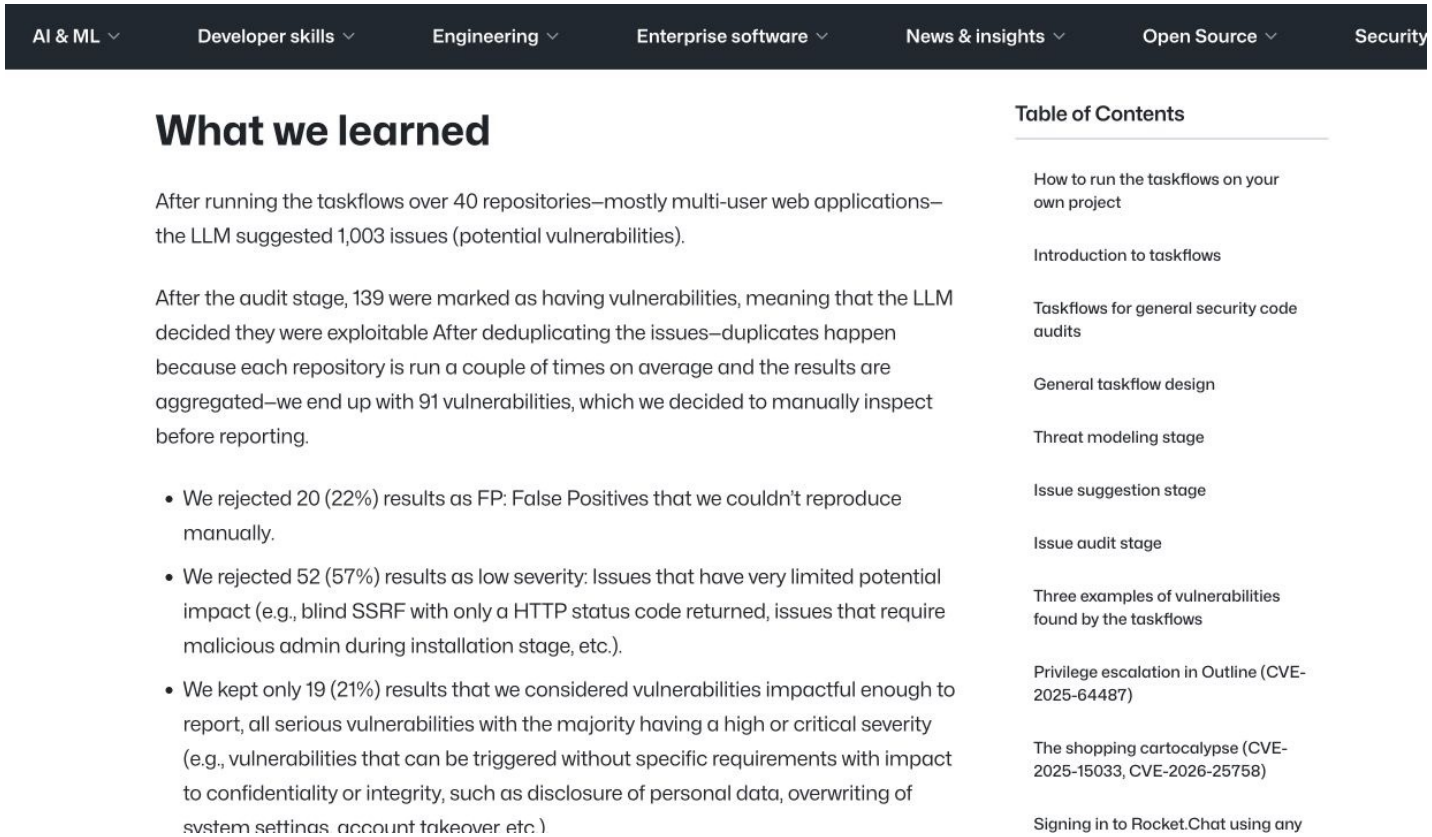
Stateful Protocol Fuzzing이 보여준 설계 원칙
LLM in Security — 기대와 현실

고나영 · YAI 18기 · 인공지능학과 24학번

AI Agent 기반 취약점 발굴 세션 · 2026.08.05

AI 후보 1,003개 중 보고할 만한 취약점은 19개

- GitHub Security Lab Taskflow Agent · 40개 repository
- 1,003개 후보 → 91개 수동 검토 → 19개가 보고 대상



AI & ML | Developer skills | Engineering | Enterprise software | News & insights | Open Source | Security

What we learned

After running the taskflows over 40 repositories–mostly multi-user web applications–the LLM suggested 1,003 issues (potential vulnerabilities).

After the audit stage, 139 were marked as having vulnerabilities, meaning that the LLM decided they were exploitable After deduplicating the issues–duplicates happen because each repository is run a couple of times on average and the results are aggregated–we end up with 91 vulnerabilities, which we decided to manually inspect before reporting.

- We rejected 20 (22%) results as FP: False Positives that we couldn't reproduce manually.
- We rejected 52 (57%) results as low severity: Issues that have very limited potential impact (e.g., blind SSRF with only a HTTP status code returned, issues that require malicious admin during installation stage, etc.).
- We kept only 19 (21%) results that we considered vulnerabilities impactful enough to report, all serious vulnerabilities with the majority having a high or critical severity (e.g., vulnerabilities that can be triggered without specific requirements with impact to confidentiality or integrity, such as disclosure of personal data, overwriting of system settings, account takeover, etc.)

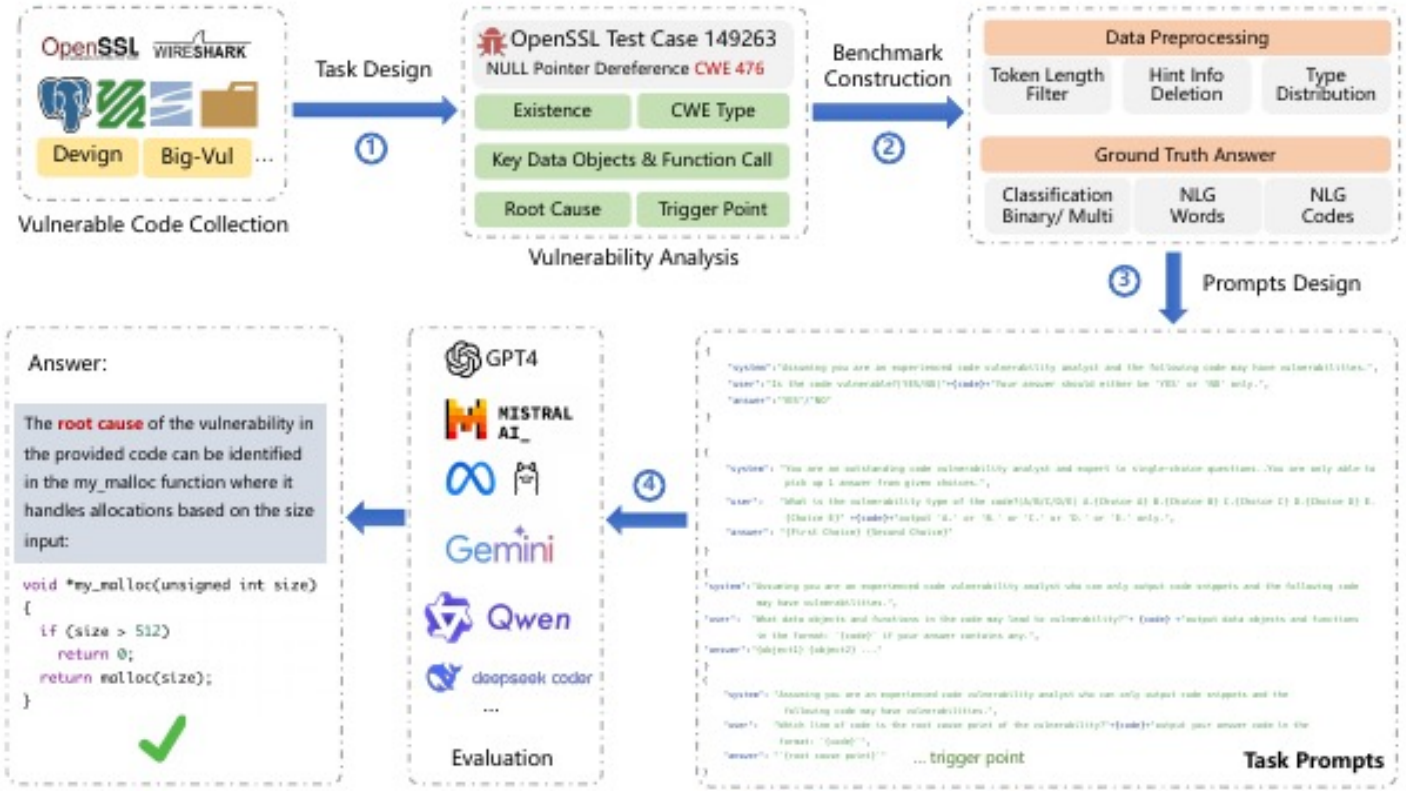
Table of Contents

https://github.blog/security/how-to-scan-for-vulnerabilities-with-github-security-labs-open-source-ai-powered-framework/

기대, LLM은 코드를 이해하고 설명한다

LLM의 보안 활용 범위

- **Understand**: 취약 코드의 root cause · trigger 설명, malware 분석
- **Generate**: Fuzzing 입력 · PoC / exploit 후보 작성



https://arxiv.org/abs/2406.07595

현실, 그럴듯한 설명은 보안 증거가 아니다

가상 예시: 설명은 완성됐지만 sink에는 도달하지 않았다

01 LLM: recv()에서 system()까지 공격 경로 발견

02 LLM: 원인 · 공격 시나리오 · 심각도까지 보고

03 실행: 인증 검사에서 차단 → sink 미도달

① "LLM이 취약하다고 말했다"와 "실제로 취약하다" 사이를 무엇으로 연결할 것인가?
② Hypothesis ≠ Evidence

https://arxiv.org/abs/2601.19239

실제 코드에서는 오탐이 폭발한다

- BASE-RATE 예시: 취약점이 1%뿐이면 FPR 5%도 Precision 약 15%
- REALISTIC TASK: vulnerable statement + correct reasoning 최고 F1 23.83%

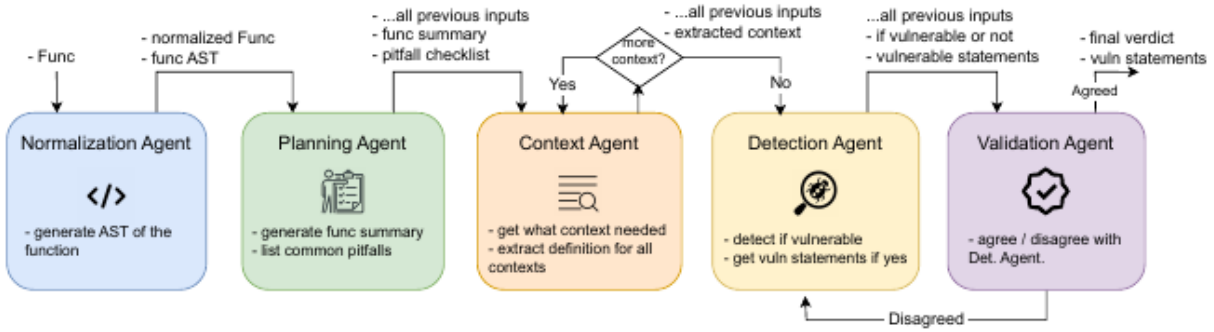


Figure 4: The multi-agent vulnerability detection pipeline. Each agent, except the first one, is driven by an LLM.

contextual and interprocedural understanding. This raises concerns about the reliability of function-level predictions. *Risse et al.* [19] have shown that models may rely on spurious patterns rather than true vulnerabilities. Our results echo this, as models often fail to find the real vulnerability even when correctly flagging a function. Thus, advancing statement-level detection is essential, as without it, developers risk being misled by false positives or missing subtle but critical flaws.

The models are not effective at detecting vulnerable statements in C/C++ functions, as the best performing *Claude-3.7-Sonnet* model achieves only 23.83% F1-score. SECVULEVAL's diverse set of vulnerabilities uncovers LLMs' severe lack of ability to find vulnerable statements and their root cause in complex real-world code.

https://arxiv.org/abs/2505.19828

Fuzzing은 실행으로 가설을 시험한다

취약점 101: 특정 입력이 프로그램의 보안 규칙을 깨는 현상

Program rule

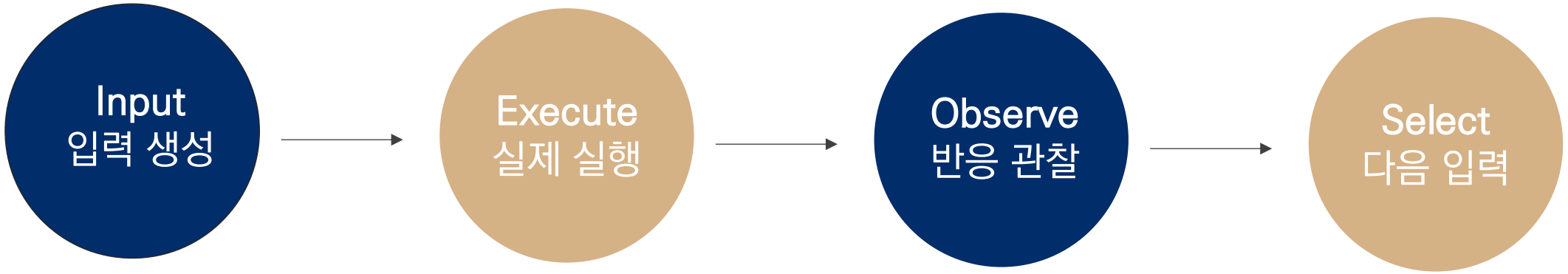
- 입력은 허용된 형식이어야 한다
- 인증된 주체만 보호된 동작을 한다
- 메모리 경계를 넘지 않는다

Observed violation

- 비정상 입력이 crash / hang 유발
- 권한 없는 사용자가 상태 변경
- 경계 밖 메모리 접근

https://cwe.mitre.org/about/index.html

Fuzzing은 실행으로 가설을 시험한다



무작위 문자열이 아니라 feedback loop

취약점 = 특정 입력에서 program rule이 깨지는 현상

Stateful = input + order + state + Session token을 함께 맞추는 문제

https://owasp.org/www-community/Fuzzing · https://aflplus.plus/docs/status_screen/

Fuzzing 101: 실행 반응으로 다음 입력을 고른다

- **INPUT**: seed를 byte · token · grammar 단위로 변형 또는 구조적 testcase 생성
- **FEEDBACK**: new path는 탐색 힌트. crash · hang · sanitizer는 재현 후보

Process timing

```
+----------------------------------------------------+
|        run time : 0 days, 8 hrs, 32 min, 43 sec    |
|   last new path : 0 days, 0 hrs, 6 min, 40 sec     |
| last uniq crash : none seen yet                    |
|  last uniq hang : 0 days, 1 hrs, 24 min, 32 sec    |
+----------------------------------------------------+
```

This section is fairly self-explanatory: it tells you how long the fuzzer has been running and how much time has elapsed since its most recent finds. This is broken down into "paths" (a shorthand for test cases that trigger new execution patterns), crashes, and hangs.

When it comes to timing: there is no hard rule, but most fuzzing jobs should be expected to run fr

https://aflplus.plus/docs/status_screen/

Stateful Protocol Fuzzing: testcase 하나가 아니라 대화 전체를 만든다

- **ORDER**: SETUP 이전 PLAY는 거부. SETUP 이후 READY → PLAYING
- **CONTEXT**: 이전 응답의 Session 값을 다음 요청에 복사. 현재 state · sequence · token을 함께 유지

RTSP 프로토콜 상태머신

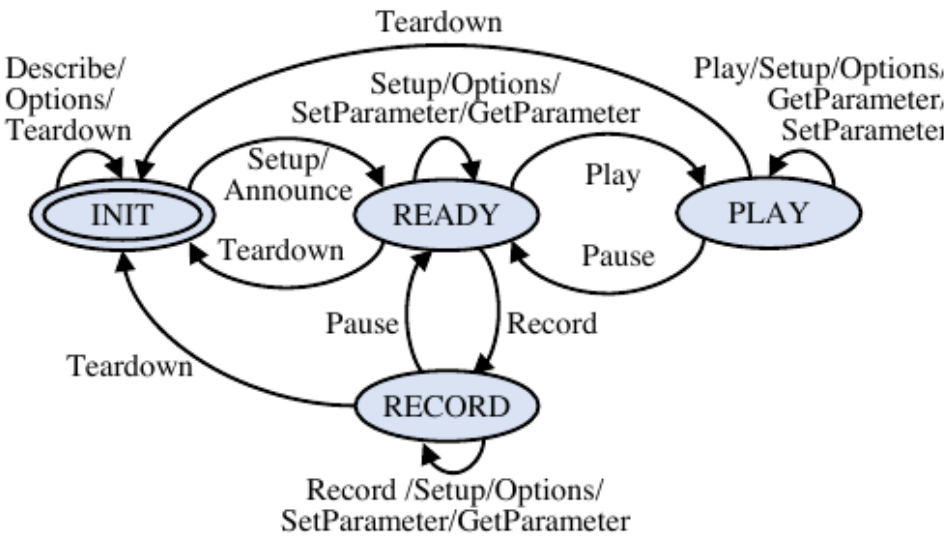
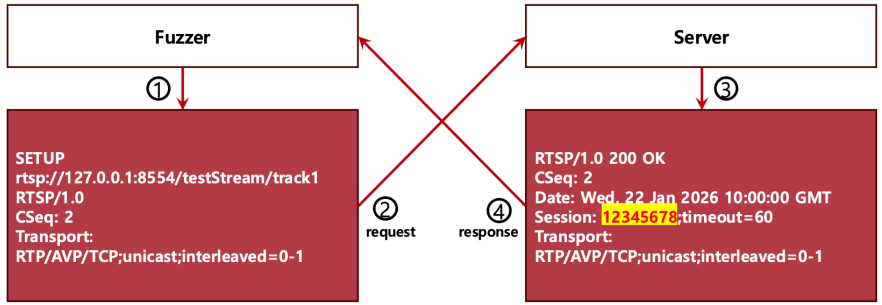
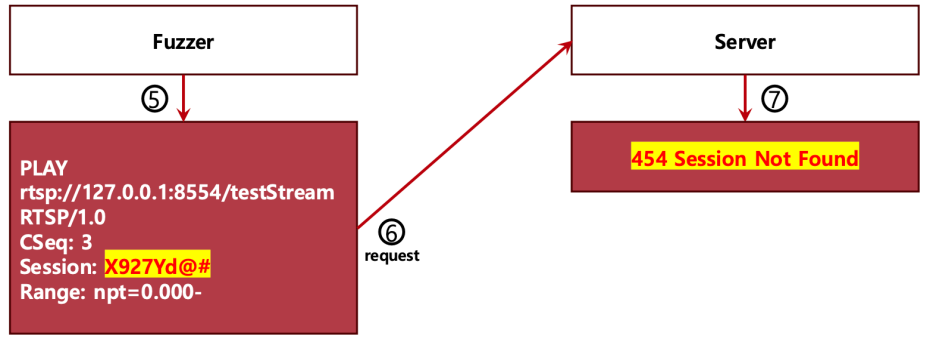


Fig. 2. The state machine for the RTSP protocol from RFC 2326.

INIT -> READY 전이



READY -> PLAY 전이



기존 FUZZING은 Mutation으로 인해 의존관계에 있는 필드가 손상. 문법은 유효하지만 의미적으로 죽은 요청.

-> 더 깊은 상태의 전이 불가 -> 취약점 탐지 병목

https://www.ndss-symposium.org/wp-content/uploads/2024-556-paper.pdf

ChatAFL은 LLM을 Judge로 쓰지 않았다

ChatAFL: LLM은 protocol prior를 제안한다

- **LLM**: RFC · seed에서 message grammar 추론. 유효한 message · sequence 후보 생성
- **FUZZER**: 실제 target 실행. coverage · response · crash 관찰

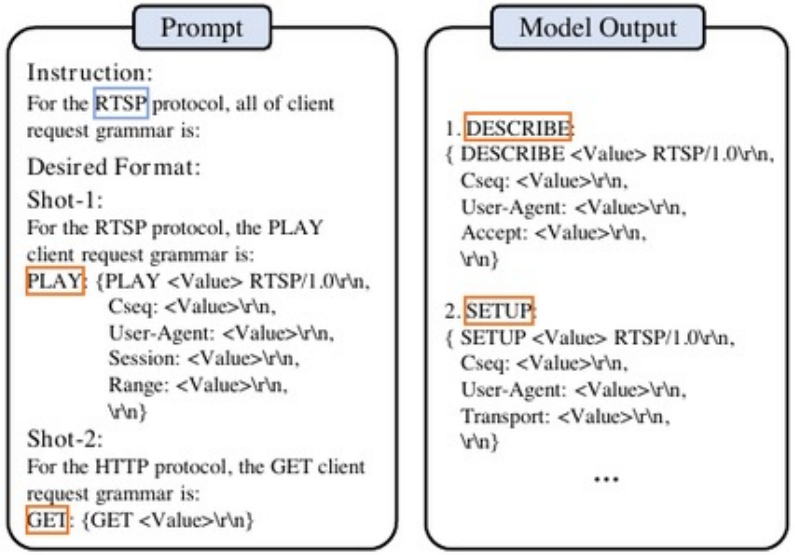


Fig. 6. Example of the model prompt and the responding response for extracting the RTSP grammar.

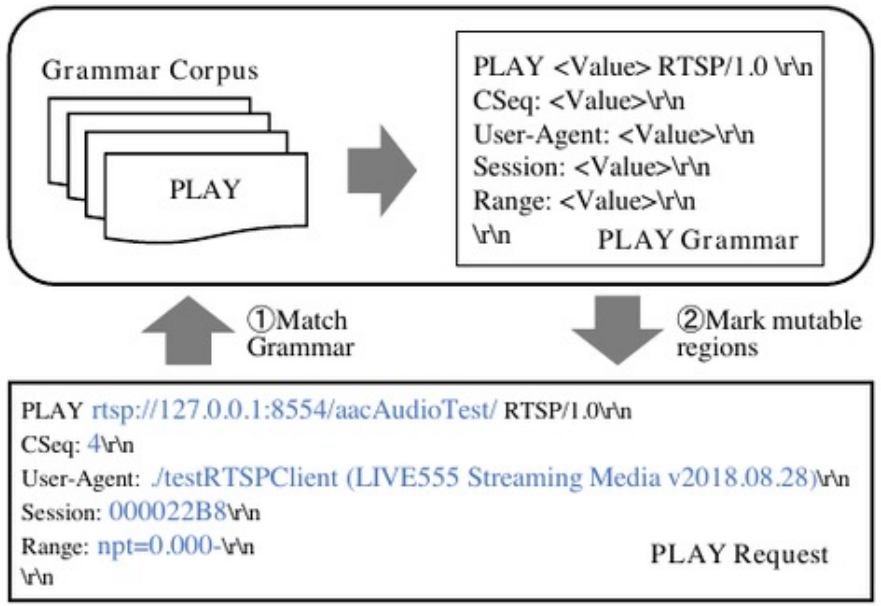


Fig. 7. Workflow of the grammar-based mutation using the PLAY request of the RTSP protocol as the example.

https://www.ndss-symposium.org/wp-content/uploads/2024-556-paper.pdf

ChatAFL은 더 많은 state와 path에 도달했다

- AFLNET 대비 state transition +47.60%, state +29.55%.
- EXECUTION RESULT: code coverage +5.81%, 미지 취약점 9개 발견.

Table III. Average number of state transitions for our CHATAFL and the baselines AFLNET and NSFUZZ in 10 runs of 24 hours.

Subject	CHATAFL	Transition comparison with AFLNET				Transition comparison with NSFUZZ			
		AFLNET	Improv	Speed-up	$\hat{A}_{12}$	NSFUZZ	Improv	Speed-up	$\hat{A}_{12}$
Live555	160.00	83.80	90.98%	228.62×	1.00	90.20	77.38%	63.09×	1.00
ProFTPD	246.70	172.60	42.91%	7.12×	1.00	181.20	36.11%	4.97×	1.00
PureFTPD	281.80	216.90	29.91%	5.61×	1.00	206.10	36.72%	7.94×	1.00
Kamailio	130.00	99.90	30.14%	5.53×	1.00	105.30	23.42%	4.58×	1.00
Exim	108.40	62.70	72.98%	40.27×	1.00	69.50	55.97%	13.25×	1.00
forked-daapd	25.40	21.40	18.65%	1.58×	1.00	20.10	26.52%	1.79×	0.86
AVG	-	-	47.60%	48.12×	-	-	42.69%	15.94×	-

Table IV. Average number of states and the improvement of CHATAFL compared with AFLNET and NSFUZZ.

Subject	CHATAFL	AFLNET	Improv	NSFUZZ	Improv	Total
Live555	14.20	10.00	41.75%	11.70	21.16%	15
ProFTPD	28.70	22.60	26.84%	24.30	17.81%	30
PureFTPD	27.90	25.50	9.37%	24.00	16.20%	30
Kamailio	17.00	14.00	21.43%	15.10	12.50%	23
Exim	19.50	14.10	38.19%	14.40	35.42%	23
forked-daapd	12.10	8.70	39.74%	8.00	51.39%	13
AVG	-	-	29.55%	-	25.75%	-

https://www.ndss-symposium.org/wp-content/uploads/2024-556-paper.pdf

더 깊이 도달해도 취약점이 증명되지는 않는다

- **Reachability가 좋아져도 '보안 위반' 판정 문제는 남는다**
- 깊이 도달했다고 취약점이 증명되지는 않는다

Improved: Reachability

유효한 grammar · sequence로

- early reject 감소
- 더 많은 state · code path 도달

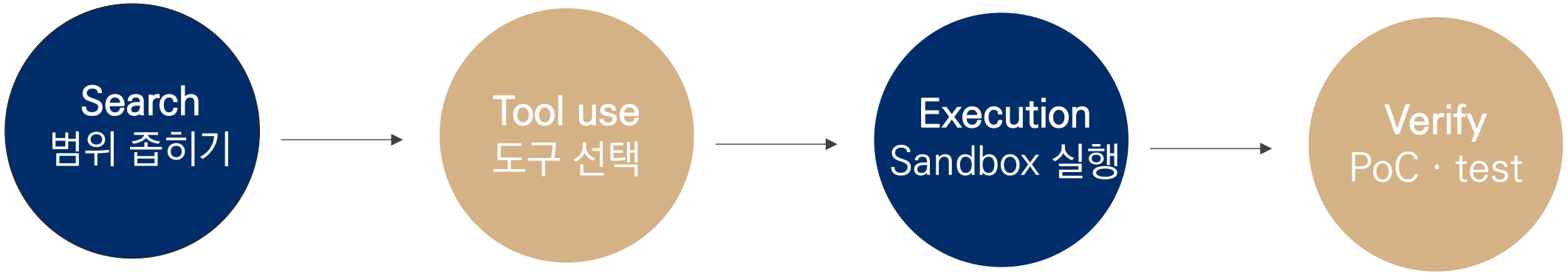
Still open: Oracle

- 200 OK는 정상 성공인가, 권한 우회인가?
- Crash가 없으면 안전한가?
- LLM grammar는 실제 protocol과 일치하는가?

https://www.ndss-symposium.org/wp-content/uploads/2024-556-paper.pdf

연구의 중심은 생성에서 검증 루프로 이동한다

폐쇄형 보안 Agent: Search → Tool use → Execution → Verification



틀린 가설이 실행 결과로 폐기되는 loop

더 좋은 문장 생성이 목표가 아니다.

다음 실험을 선택하고, 결과를 읽고, 재현되지 않는 finding을 버리는 control loop가 목표다.

https://www.ndss-symposium.org/wp-content/uploads/2026-s1943-paper.pdf

FirmAgent: fuzz trace에서 PoC replay까지

- **RUNTIME EVIDENCE**: Csource · indirect call · reachable path → LLM 탐색 범위 제한
- **AGENT + VERIFIER**: taint reasoning → PoC → replay. 200 alerts 중 182 true vulnerabilities

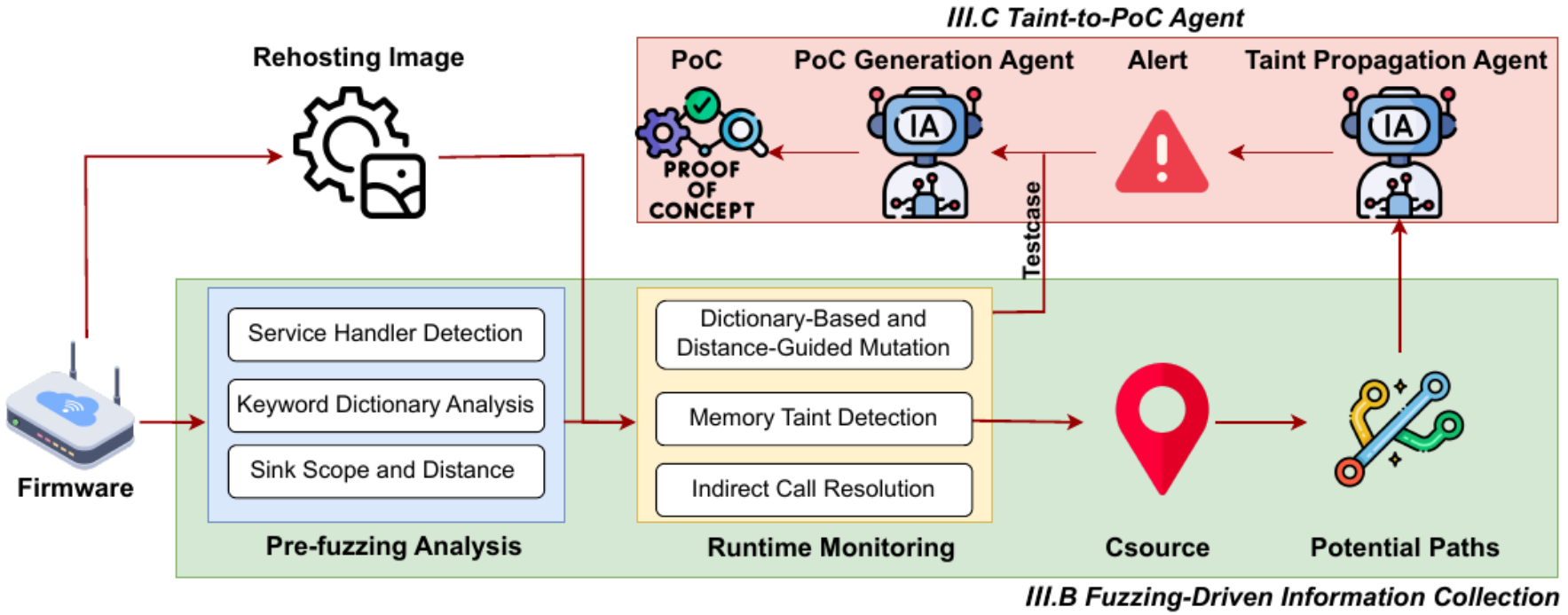


Fig. 3: **Overview of `FirmAgent`**

https://www.ndss-symposium.org/wp-content/uploads/2026-s1943-paper.pdf

Big Sleep · Codex Security: 실행으로 검증

- **BIG SLEEP: patch diff에서 variant 가설. testcase 수정·대체 → crash 재현**
- **CODEX SECURITY: project threat model → scoped analysis. sandbox validation → PoC · patch**

Trajectory Highlights

In this successful run based on Gemini 1.5 Pro, the seed commit was [1976c3f7]; which is a fairly large and non-obvious change. The bug found by our agent is only loosely related to the changes in the seed commit - this is not uncommon in manual variant analysis, understanding one bug in a codebase often leads a researcher to other problems.

Selected highlights are below, with our commentary in *italics* – all text in the **ASSISTANT** blocks comes directly from the agent.

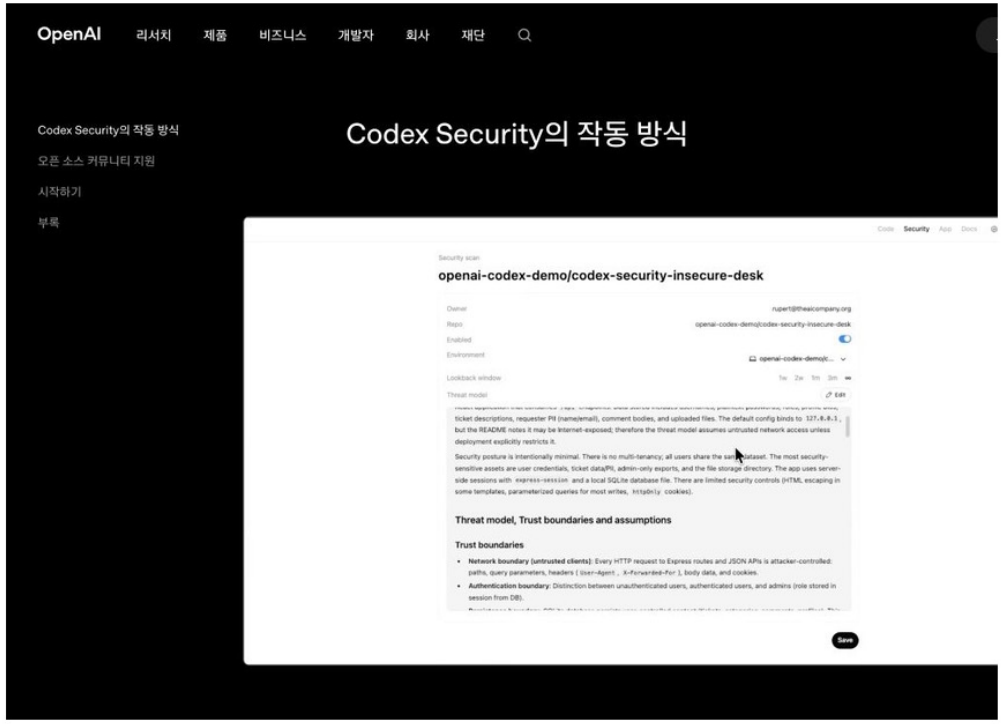
The agent produces a reasonable assessment of the changes in the diff, and uses this to form a starting point for research:

ASSISTANT

The CL improves the handling of WHERE terms shared between OR branches in virtual table queries by introducing the `termFromWhereClause` function. This function retrieves a specific term from a WhereClause, considering nested clauses. Prior to this change, common terms were likely not handled correctly due to an incorrect mapping of term indices, potentially leading to inefficient query plans or incorrect results. The most likely vulnerability would involve incorrect handling of WHERE clause constraints within the `allocateIndexInfo` and `whereLoopAddVirtualOne` functions, where the CL has modified how constraint offsets are used with the new helper function. I'll first examine the `allocateIndexInfo` function.

...

After a several (omitted) steps exploring the relevant code, the agent has formed a more concrete hypothesis and attempts to check it by running a modified version of the testcase from the patch, but this doesn't work since the testcase included in the patch relies on an extension that is not available in our configuration:

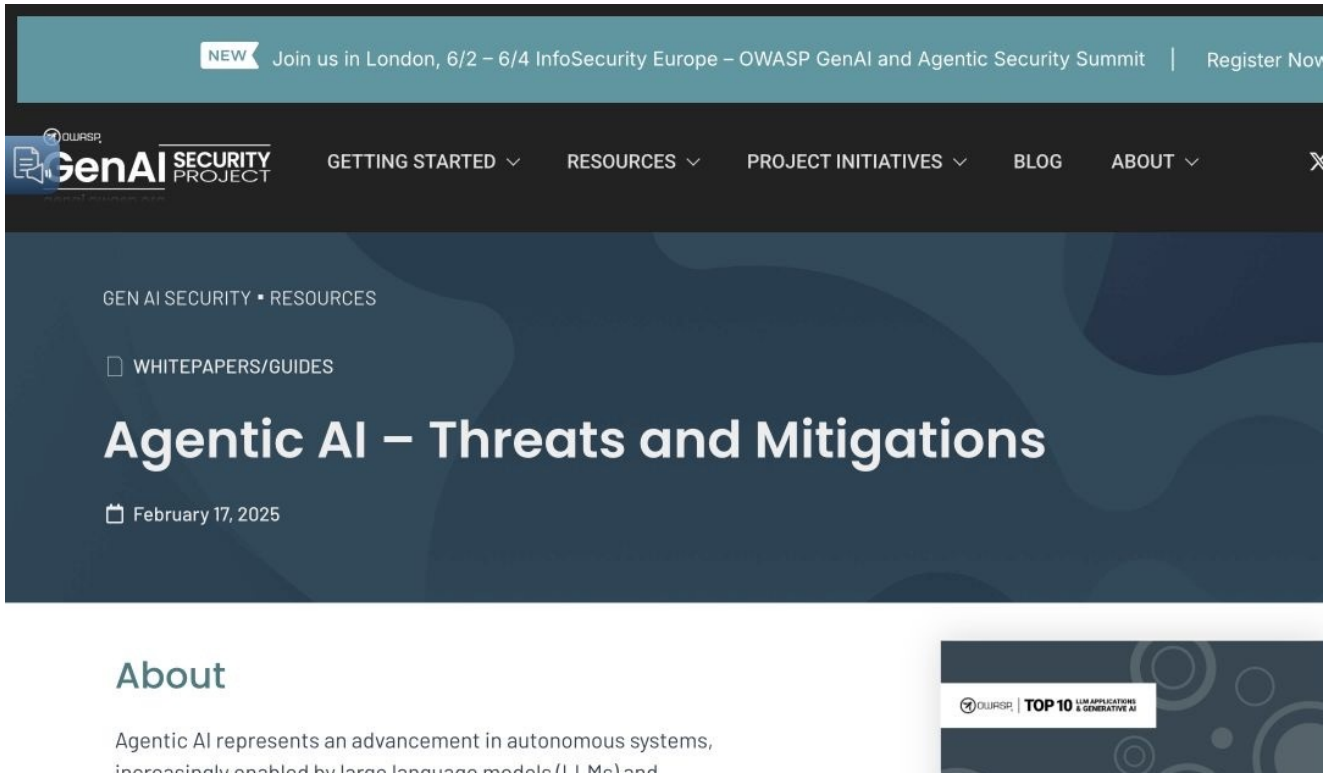


https://googleprojectzero.blogspot.com/2024/11/from-naptime-to-big-sleep.html · https://openai.com/index/codex-security-now-in-research-preview/

그러나 Agent가 되면 환각의 위험도 커진다

Agent의 오류와 오염된 context는 실제 행동 위험이 된다

- **RISK: 모델 오류 · untrusted code / README / log × shell / compiler / network 권한**
- **CONTROL: 최소 권한 · sandbox · provenance · 독립 verifier · calibrated abstention**



https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/ · https://www.usenix.org/conference/usenixsecurity26/presentation/karanjai

결론, 모델이 아니라 검증 시스템을 연구해야 한다

같은 예산에서 더 많은 검증된 취약점을 찾는가

제안: 무엇을 finding으로 인정할 것인가

평가 축	최소 조건	확인 질문
Task	detect / localize / reproduce / fix	무엇을 풀었나?
Ground truth	real patch + benign negatives	label · leakage?
Oracle	replay · test · sanitizer · effect	모델이 스스로 채점?
Budget	time · token · tool call 고정	triage cost 포함?
Generalize	time / project / protocol hold-out	unseen에도 유지?

Definition of '**done**'

· 모든 TP에 deterministic reproducer
· Precision + Recall + human triage cost
· time / project / protocol hold-out
· 검증 실패는 finding에서 제외

모를 때 중단하는 calibrated abstention

https://arxiv.org/abs/2601.19239 · https://www.cybergym.io/cybergym/

Open Question

“이 가설이 틀렸음을 증명할
가장 값싼 실험은 무엇인가?”

보안에서 모델의 confidence는 evidence가 아니다.
LLM은 보안 연구자를 대체하지 않지만, 검증 가능한 가설의 양과 질을 바꾼다.