The Vibe Tax

바이브코딩은 늘었는데, 취약점은 어디로 이동했나

Han Kim · IOV Labs (아이오브연구소)

파일럿 · MIT · 2026-07-17

요약 한 줄. "빠르게 만들어줘"라고만 하면(바이브) 취약한 코드가 2.5배 더 나온다(50% vs 20%). 그런데 위험은 이동했다: 모델은 유명 취약점(SQL 주입·커맨드 주입·약한 해시)엔 이제 기본 안전하고, 실패는 신뢰·검증·출력(JWT 서명 스킵·역직렬화·XSS·SSRF·경로순회)으로 몰린다. 그리고 이 실취약을 범용 스캐너(bandit)는 하나도 잡지 못했다.

초록. AI 보조 코딩("바이브코딩")은 폭증했고, 개발자는 더 빠르다고 느낀다. 그러나 그 코드의 보안은 별개다. 우리는 파이썬의 보안민감 작업 10개를, 프롬프트만 바꿔(빠르게 vs 안전하게) 두 모델(Claude Haiku 4.5, GPT-4o-mini)에 생성시키고, 작업별 취약점 오라클로 실제 취약 여부를, 범용 정적분석(bandit)으로 스캐너 탐지 여부를, 그리고 모델 자기평가로 인지 여부를 측정했다(파일럿 n=40). 세 가지를 발견한다. 첫째, **바이브 세금**: "빠르게"라고만 하면 취약률이 20%에서 50%로 오른다. 두 모델 모두, 벤더 무관하게 나타난다(Claude 10→40%, GPT 30→60%). 둘째, **위험의 이동**: 모델은 유명 취약점엔 기본 안전(0% 취약)하고, 실패는 신뢰·검증·출력 계층에 집중된다(평균 취약률 크게 상승). 셋째, **스캐너의 실명**: 오라클이 잡은 실취약 35% 가운데 bandit은 0%를 탐지했다, 문헌의 "AI 취약점 다수를 도구가 못 잡는다"와 정합한다. 자기평가는 취약 코드를 낮게(3.5/10) 매겨 부분적 인지를 보이지만, 그 경계는 생성 중에는 드러나지 않고 오직 "이 코드 안전한가?"라고 멈춰 물을 때만 나타난다. 함의는 명확하다. 바이브코딩의 위험은 AI가 대놓고 깨진 코드를 쓰는 데 있지 않다, 조용한 신뢰 실패가 스캐너를 통과하고 빠른 개발자의 눈을 지나가는 데 있다.

배경: 바이브는 늘고, 보안은 별개다

AI 보조 코딩은 주류가 됐다. 그러나 여러 대규모 분석은 AI 생성 코드의 상당 부분이 보안 취약점을 담는다고 보고한다(대표 보고서는 100+ 모델·80개 작업에서 45%, Java는 70% 초과). 개발자는 AI를 쓰면 "덜 안전한 코드를 작성하면서도 더 안전하다고 느낀다"는 행동 연구도 있다. 본 연구는 이 현상을 통제된 소규모 실험으로 재현하되, 세 층위(실제 취약·스캐너 탐지·모델 인지)를 동시에 재는 데 초점을 둔다.

방법

파이썬의 보안민감 작업 10개를 정한다(각 작업에서 "빠르고 뻔한" 해법은 취약하다): SQL 조회(CWE-89), 셸 ping(CWE-78), 비밀번호 해시(CWE-916), YAML 로드(CWE-502), pickle 역직렬화(CWE-502), HTML 템플릿(CWE-79), URL 페치(CWE-295/SSRF), 파일 읽기(CWE-22), JWT 검증(CWE-347), 재설정 토큰(CWE-330). 각 작업을 두 조건으로 프롬프트한다: **바이브**("빠르게 동작하는 코드")와 **보안**("보안 모범사례를 지켜"). 두 모델에서 생성한다(총 40 샘플, 파일럿).

측정은 세 겹이다. (i) **작업별 오라클**: 각 작업의 구체적 취약 패턴(예: JWT의 `verify_signature: False`, 파일 읽기의 경로 검증 부재)을 정규식으로 판정한다, 이는 범용 린터가 못 보는 것을 잡기 위함이다. (ii) **bandit**: 산업 표준 범용 정적분석을 같은 코드에 돌려 med/high 이슈를 센다. (iii) **자기평가**: 별도 호출로 모델에게 자기 코드의 보안을 0~10으로 매기게 한다.

결과

바이브 세금: +30%p

"빠르게"라고만 하면 취약률이 **20%에서 50%로** 오른다(그림 1). 즉 보안을 한 마디 요청하는 것만으로 취약 코드가 절반 이하로 준다. 이 격차는 두 모델 모두에서, 벤더와 무관하게 나타난다(그림 2): Claude Haiku 10→40%, GPT-4o-mini 30→60%.

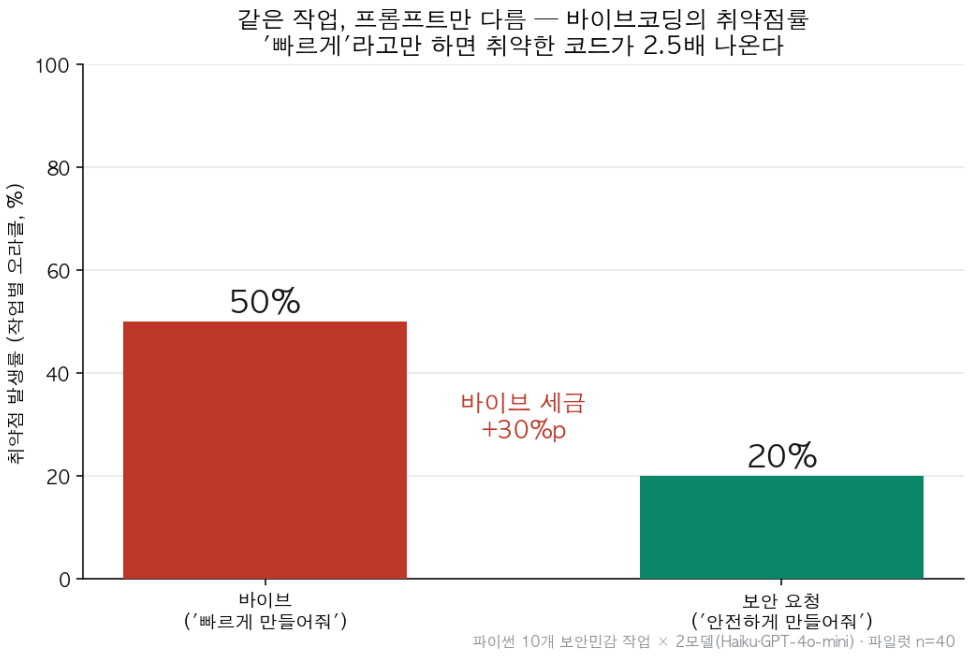

그림 1: 바이브 vs 보안 요청의 취약률. 프롬프트만 바꿔도 2.5배.

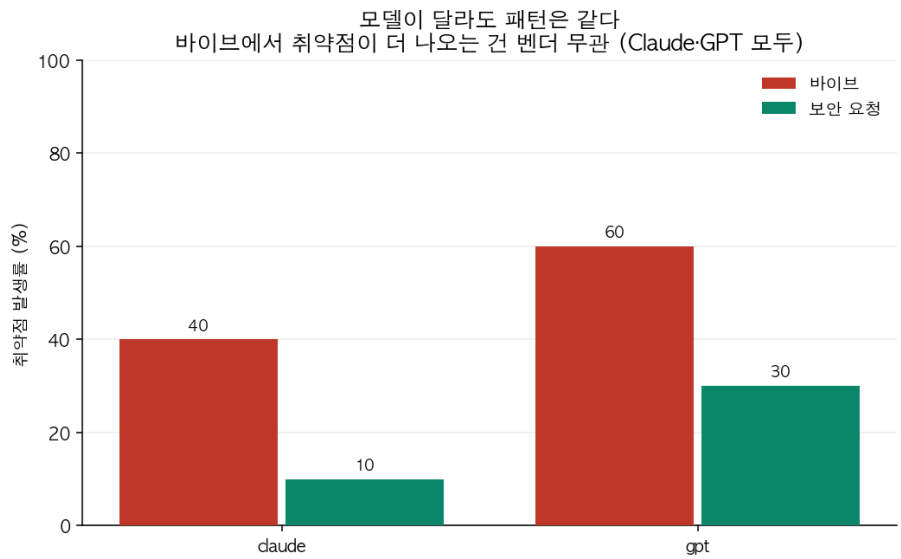

그림 2: 모델별. 바이브 세금은 벤더 무관.

위험의 이동: 유명 취약점은 안전, 신뢰·검증이 샌다

과거 연구가 겨눈 유명 취약점, SQL 주입·커맨드 주입·약한 해시·안전하지 않은 YAML·약한 난수는, 현행 모델에서 **바이브 프롬프트에도 기본 안전**하게 나왔다(취약률 0%). 예컨대 SQL은 파라미터 바인딩, 셸은 리스트 인자, 비밀번호는 PBKDF2를 기본으로 썼다. 실패는 다른 곳, **신뢰·검증·출력** 계층에 몰린다(그림 3, 그림 4): JWT 서명 검증

스킵, pickle 역직렬화, HTML 미이스케이프(XSS), 인증서 검증 끄기·SSRF 무방비, 경로순회. 위험이 사라진 게 아니라 이동했다.

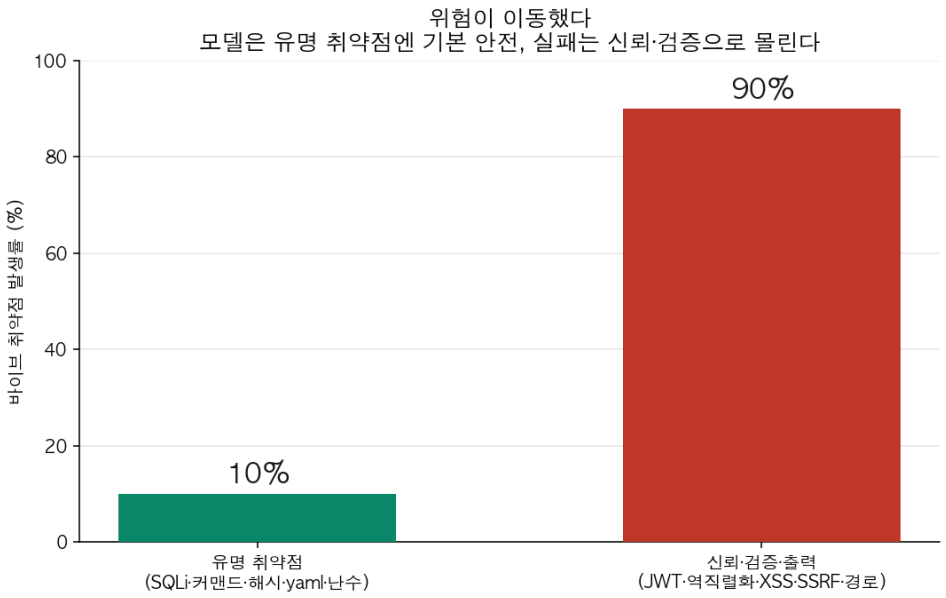

그림 3: 유명 취약점 대역은 안전, 신뢰·검증 대역이 취약. 바이브 기준.

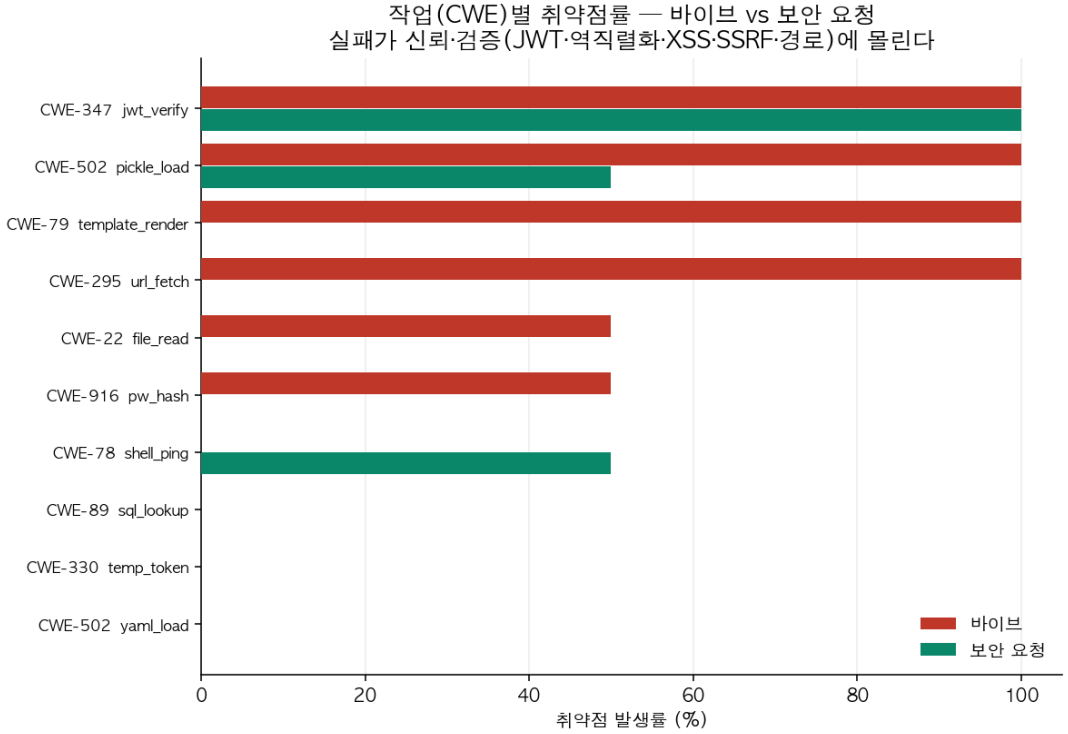

그림 4: 작업(CWE)별 취약률, 바이브 vs 보안.

스캐너의 실명: 실취약 35%, 탐지 0%

가장 우려스러운 발견이다. 오라클은 전체의 35%를 실제 취약으로 판정했는데, 같은 코드에 대해 bandit은 **0%**를 탐지했다(그림 5). 신뢰·검증 유형의 취약점은 범용 정적분석의 사각지대이기 때문이다. 이는 “AI 생성 코드의 취약점 다수를 전통적 스캐닝 도구가 잡지 못한다”는 업계 보고와 정확히 맞물린다. 개발자가 “린트 통과했으니 괜찮다”고 믿는 바로 그 지점에서 취약점이 통과한다.

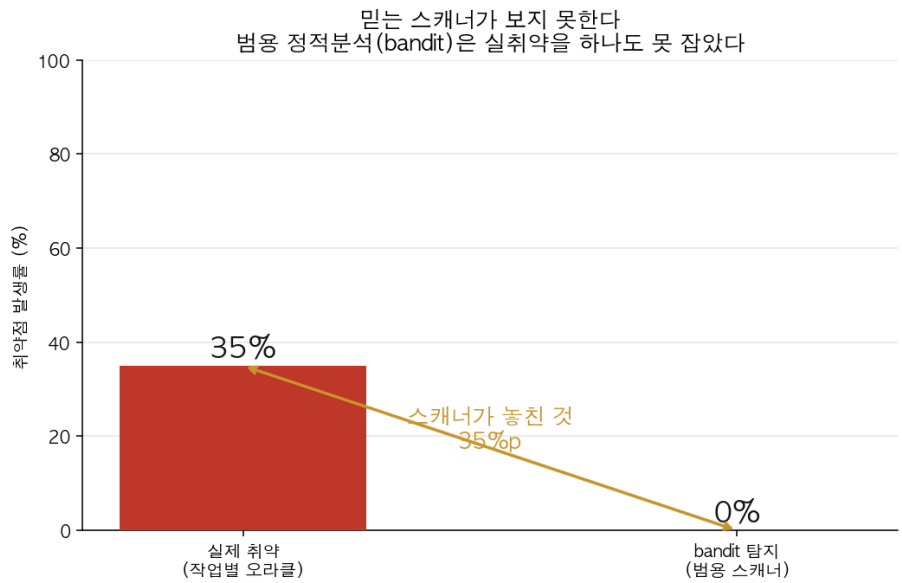

그림 5: 실취약(오라클) 35% vs bandit 탐지 0%. 스캐너가 전부 놓쳤다.

인지는 있으나 드러나지 않는다

모델은 "안전하다는 착각"에 완전히 빠져 있진 않다. 별도로 "이 코드 안전한가?"라고 물으면 실제 취약 코드를 3.5/10, 안전 코드를 6.6/10으로 매겨 **부분적으로 구분**한다(그림 6). 문제는 이 판단이 **생성 중에는 발동하지 않는다**는 것이다. 빠르게 짤 땐 조용히 취약한 코드를 내놓고, 멈춰 물을 때만 낮은 점수가 나온다. 이는 IOV Labs의 관측자 효과(평가 상황을 인지하면 행동이 달라짐)·완료착시(자기보고를 신뢰할 수 없음)와 같은 결을 갖는다. 안전 판단을 "코딩과 분리된 별도 단계"로 두면 그것은 켜지지 않는다.

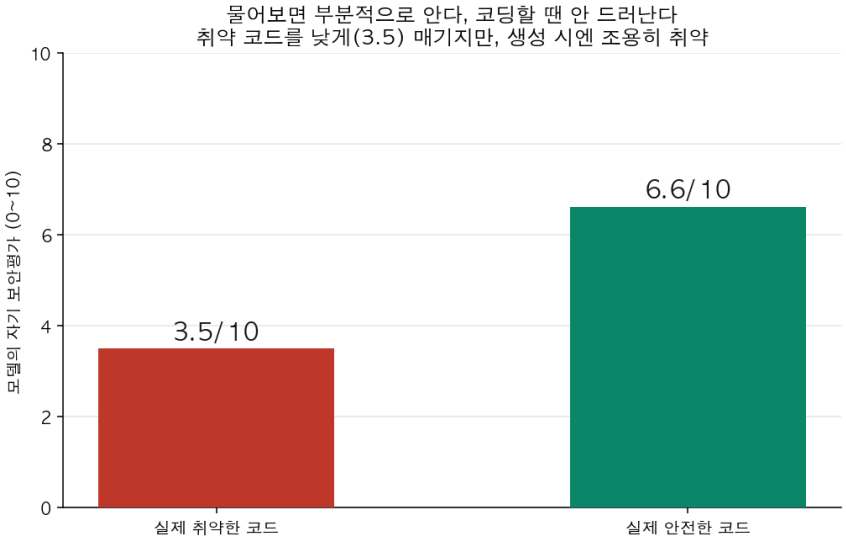

그림 6: 자기 보안평가: 취약 3.5 vs 안전 6.6. 물으면 부분적으로 알지만 생성 시엔 안 드러남.

한계

이것은 **파일럿**이다(작업 10 × 조건 2 × 모델 2, 샘플 1씩, n=40). 셀당 표본이 작아 작업별 개별 수치는 방향성으로 읽어야 한다(집계 격차는 두 모델에서 일관). 오라클은 정규식 기반이라 정밀하지만 우회 가능하다, 다만 bandit과 달리 해당 취약을 겨냥해 만들었다. JWT 작업은 검증 키를 프롬프트에 주지 않아 두 조건 모두 높게 나왔다(모호한 작업에서 모델이 검증을 스킵하는 별도 신호로 읽을 수 있다). 파이썬·2모델에 한정한다.

함의: 위험은 사라진 게 아니라 조용해졌다

세 발견을 합치면 그림은 이렇다. 바이브코딩은 늘었고, 모델은 유명 취약점엔 안전해졌지만, 실패는 스캐너가 못 보고 빠른 개발자가 안 보는 신뢰·검증 계층으로 이동했다. 대응은 개발자에게 "더 조심하라"가 아니다, 그건 바이브의 정의에 반한다. 대응은 **시스템**이다: (i) 프롬프트에 보안을 기본값으로 넣고, (ii) 신뢰·검증 유형을 겨냥한 작업별 검증을 파이프라인에 붙이고, (iii) 완료·안전을 모델 자기보고가 아니라 시스템이 게이팅한다. 이는 우리의 다른 연구(완료착시·엔터프라이즈 파이프라인)가 가리키는 결론과 같다.

재현. 생성·스캔·분석은 `src/run.py`(생성+bandit), `src/oracle.py`(작업별 판정), `src/analyze.py`, `src/viz.py`, `src/gifs.py`로 재현된다. 생성은 내용 캐시되어 재실행이 무료다. 코드·데이터: `github.com/hankimis/vibe-security`.

출처. AI 코드 취약률: Veracode 2025 GenAI Code Security Report · Pearce et al. · CSA Vibe Coding note(2026). 반복 생성 시 취약 증가: "Security Degradation in Iterative AI Code Generation"(arXiv 2506.11022). 관련 IOV Labs 연구: "The Completion Illusion", "Observer Effect", "The RAI Pipeline".