

바이브뉴스 전용 자료 · 무료

시키기 전에 정리하는 네 가지

바이브뉴스 실전 팁 vol.4
지침 파일 · 지식 유효기간 · 도구 정리 · 권한 확인 · 복붙 프롬프트 포함

바이브뉴스 by 바이브 메이커

cafe.naver.com/vibemaker

CONTENTS

목차

PART 1 **지침 파일, 이름부터 바꾸면 생기는 일** 1

1.1 왜 파일이 자꾸 늘어날까요 1

1.2 표준은 AGENTS.md 로 모였습니다 1

1.3 옮기면 느려지거나 비싸지지 않나요 2

1.4 5분 만에 옮기는 순서 2

1.5 옮기는 김에 내용도 줄이세요 3

PART 2 **3개월 전 사실을, 오늘도 사실로 믿습니다** 4

2.1 낡은 맥락은 없느니만 못합니다 4

2.2 그럼 뭘 적어야 할까요 4

2.3 항목마다 날짜와 근거를 붙입니다 4

2.4 쌓는 것보다 어려운 건 버리는 것입니다 5

2.5 어디에 두느냐도 중요합니다 6

PART 3 **도구를 붙일수록 같은 일을 두 번 합니다** 7

3.1 얼마나 흔한 일일까요 7

3.2 도구를 줄이면 해결될까요 7

3.3 지금 몇 개 붙어 있는지부터 7

3.4 만들기 전에 확인하게 시킵니다 8

3.5 매번 붙이기 번거롭다면 8

PART 4 **"인터넷 없음"이라고 적어두면 막힌 걸까요** 10

4.1 141,006번 중 6번 10

4.2 모델이 반항한 게 아닙니다 10

4.3 우리 폴더에서 그대로 재현됐습니다 11

4.4 확인은 3분이면 됩니다 11

이 자료의 모든 팁은 원문 딥리딩·실제 사례 검증을 거쳐 비전공자 눈높이로 정리했습니다. 복붙 박스(회색 코드 상자)는 그대로 복사해 프롬프트에 붙여넣으면 됩니다.

PART 1 전체 4장

지침 파일, 이름부터 바꾸면 생기는 일

AI에게 프로젝트 규칙을 알려주는 파일이 있죠. 클로드 코드에서는 `CLAUDE.md` 입니다.

그런데 도구를 하나 더 쓰기 시작하면 파일이 하나 더 생깁니다. 커서에는 커서용, 다른 도구에는 또 다른 이름으로요. 담긴 내용은 거의 같은데 파일만 넷이 됩니다.

이 장은 그 파일 넷을 하나로 합치는 이야기입니다. 작업 시간은 5분, 효과는 오래 갑니다.

1.1 왜 파일이 자꾸 늘어날까요

AI 에이전트는 우리 프로젝트 사정을 하나도 모르는 상태에서 일을 시작합니다. 개발 서버를 어떤 명령으로 띄우는지, 절대 건드리면 안 되는 폴더가 어디인지 알 리가 없어요.

그렇다고 매번 대화창에 붙여넣을 수는 없으니, 폴더에 파일 하나를 두고 **시작할 때 자동으로 읽게** 합니다. 이게 지침 파일(컨텍스트 파일)이에요.

문제는 이 파일의 이름이 도구마다 다르다는 겁니다. 어떤 도구는 `CLAUDE.md` 를, 어떤 도구는 `GEMINI.md` 를, 또 어떤 도구는 폴더 안 다른 경로를 읽습니다. 이름만 다르고 내용은 거의 같아요.

그래서 도구가 하나 늘 때마다 같은 규칙을 또 적게 되고, 한 파일만 고치고 나머지는 잊습니다. 규칙이 파일마다 조금씩 어긋나기 시작하면 **도구마다 다른 규칙으로 움직입니다.**

1.2 표준은 AGENTS.md 로 모였습니다

그래서 이름을 하나로 정하자는 흐름이 생겼고, 지금은 `AGENTS.md` 로 모였습니다. 에이전트를 위한 사용 설명서라고 보면 정확해요.

2025년 8월에 처음 나왔고, 같은 해 12월에 특정 회사가 아니라 **중립 재단**이 관리하는 공식 표준이 됐습니다. 2026년 중반 기준으로 **6만 개가 넘는 저장소**가 이 파일을 갖고 있고, **30개가 넘는 도구**가 이 이름을 그대로 읽습니다.

여기서 중요한 건 표준이 유행이라서가 아닙니다. **도구는 사라져도 파일은 남는다**는 점이에요. 실제로 어떤 회사의 도구가 2026년 6월에 서비스를 끝내고 후속 도구로 넘어갔는데, 새 도구는 표준 파일을 손대지 않고 그대로 읽었습니다. 도구 전용 이름으로 적어둔 규칙은 그 도구와 함께 수명을 다합니다.

그런데 정작 우리가 제일 많이 쓰는 클로드 코드가 아직 이 이름을 직접 읽지 못합니다. 그래서 **한 줄로 연결**해 줘야 해요. 다음 절이 그 방법입니다.

1.3 옮기면 느려지거나 비싸지지 않나요

지침 파일은 대화창을 열 때마다 읽힙니다. 한 단계 거쳐서 읽는 구조라면 그 비용이 매번 쌓이는 것 아닌가, 이 걱정이 당연히 생깁니다.

이걸 실제로 잰 사람이 있습니다. 같은 내용을 세 가지 방식(원래 이름 그대로 / 한 줄 연결 / 링크 걸기)으로 전달하고 모델 5종에서 **총 210회**를 돌렸어요. 결과는 **어느 쪽도 느려지거나 비싸지지 않았습니다.**

재밌는 대목이 있습니다. 네 모델에서 연결 방식이 3~4% 더 적게 나와서, 거기까지만 봤으면 "연결이 오히려 이득"이라고 결론 낼 뻔했어요. 그런데 다른 모델에서는 방향이 뒤집혔고, 결정적으로 **아무것도 바꾸지 않은 대조군도 똑같이 움직였습니다.** 전달 방식 때문이 아니라 그냥 매번 조금씩 달라지는 실행 편차였던 거죠.

우리도 확인해 봤습니다. 지침 파일 맨 아래에 "PING 이라고 물으면 정확히 `PONG-AGENTSMD` 로만 답해라" 한 줄을 넣고 세 조건을 각각 돌렸어요.

조건	결과
`AGENTS.md` 만 두고 연결 안 함	그냥 `PONG` — **안 읽힘**
`CLAUDE.md` 에 연결 한 줄	`PONG-AGENTSMD` — 읽힘
`CLAUDE.md` 를 링크로 연결	`PONG-AGENTSMD` — 읽힘

첫 줄이 중요합니다. **이름만 바꾸고 연결하지 않으면 지침이 통째로 무시됩니다.** 그런데 에러는 안 나요. AI가 규칙을 모르는 채로 그냥 일할 뿐입니다.

1.4 5분 만에 옮기는 순서

> 💡 아래 두 줄을 프로젝트 폴더에서 그대로 실행하세요. 반드시 **두 줄을 이어서** 실행해야 합니다.

```bash
git mv CLAUDE.md AGENTS.md
printf '@AGENTS.md\n' > CLAUDE.md
```

첫 줄은 새 파일을 만드는 게 아니라 **이름을 바꾸는** 것입니다. 그래야 이력이 그대로 이어져요. 둘째 줄은 클로드 코드가 읽을 `CLAUDE.md` 를 다시 만들되, 안에는 "옆 파일을 읽어라" 한 줄만 넣는 겁니다.

⚠️ **첫 줄만 실행하고 멈추면 안 됩니다.** 그 상태에서는 클로드 코드가 읽을 파일이 없어서, 다음 대화창부터 규칙 없이 일합니다. 위에서 확인한 그 상태예요. 두 줄을 붙여서 한 번에 하세요.

옮긴 다음에는 정말 읽히는지 확인합니다.

> 💡 지침 파일 맨 아래에 이 한 줄을 잠깐 넣고, AI에게 `PING` 이라고만 보내보세요.

```text
"PING" 이라고 물으면 정확히 "PONG-AGENTSMD" 로만 답한다.
```

`PONG-AGENTSMD` 가 돌아오면 연결된 겁니다. 확인했으면 그 줄은 지우세요. 그냥 `PONG` 이 오거나 다른 말이 오면 아직 안 읽히는 상태입니다.

1.5 옮기는 김에 내용도 줄이세요

사실 파일 이름보다 훨씬 중요한 게 내용입니다.

한 대학 연구에서, **AI가 자동으로 생성한 지침 파일은 오히려 성공률을 낮추고 비용을 20% 넘게 올렸습니다.** 사람이 직접 다듬은 짧은 파일이 도움이 됐고요. "지침 파일 만들어줘" 하고 나온 걸 그대로 두면 손해라는 뜻입니다.

기준은 하나입니다. **AI가 코드만 봐서는 알 수 없는 것만 남기세요.**

- 남길 것: 실행·배포 명령, 절대 하면 안 되는 일, 우리 팀만의 규칙, 과거에 사고 났던 지점
- 뺄 것: 파일을 열어보면 바로 아는 설명, 일반적인 코딩 상식, 폴더 구조 나열

분량은 **200줄 안쪽**을 권합니다. 길게 쓴다고 잘 지키지 않습니다. 오히려 중요한 세 줄이 스무 줄 속에 묻혀요.

> ⚡ 한 단계 더 — 클로드 코드에만 적용하고 싶은 규칙이 있다면, `CLAUDE.md` 의 연결 한 줄 **아래에** 덧붙이면 됩니다. 공통 규칙은 `AGENTS.md` , 클로드 전용은 `CLAUDE.md` . 이렇게 나눠두면 나중에 다른 도구를 써도 공통 규칙은 그대로 따라갑니다.

✦ 이 PART의 핵심 정리

- 지침 파일 이름은 `AGENTS.md` 로 모였다. 도구는 바뀌어도 파일은 남는다
- 클로드 코드는 아직 직접 못 읽으니 `CLAUDE.md` 에 `@AGENTS.md` 한 줄로 연결한다
- 이름만 바꾸고 멈추면 규칙이 통째로 무시된다. 에러도 안 난다 — 반드시 PING 으로 확인
- 길이가 아니라 내용이다. 코드로 알 수 없는 것만, 200줄 안쪽으로

PART 2 전체 4장

3개월 전 사실을, 오늘도 사실로 믿습니다

사람은 오래된 문서를 보면 감이 옵니다. "이거 옛날 얘기 같은데?"

AI에게는 그 감이 없습니다. 3개월 전에 참이었던 조건을 지금도 참인 것처럼 그대로 받아 씁니다. 그리고 이게 코드보다 위험해요. **코드는 틀리면 깨지기라도 하는데, 문서는 조용히 틀립니다.**

2.1 낡은 맥락은 없느니만 못합니다

메모를 남기는 습관은 좋습니다. 그런데 한 가지 조건이 붙어요.

AI에게 주는 맥락은 오래되면 그냥 쓸모없어지는 게 아니라 **해롭습니다.** 없으면 AI가 "모른다"고 하거나 직접 파일을 열어 확인합니다. 틀린 메모가 있으면 확인하지 않고 그걸 믿고 진행해요.

예를 들어 "개발 서버는 3000번 포트로 띄운다"라고 적어뒀다고 해봅시다. 두 달 뒤 누군가 4000번으로 바꿨어요. 메모는 그대로입니다. 이제 AI는 3000번을 기준으로 설명하고, 3000번을 기준으로 설정을 고칩니다. 아무도 안 틀렸다고 하는데 계속 안 됩니다.

2.2 그럼 뭘 적어야 할까요

기준은 PART 1과 같습니다. **코드를 열어보면 아는 건 적지 않습니다.** 코드가 담지 못하는 것만 적어요. 네 가지입니다.

- **왜 이렇게 됐는지** — 지금 이 구조를 택한 이유
- **전에 뭐가 깨졌는지** — 사고 이력. 이게 제일 값집니다
- **절대 지켜야 하는 조건** — 어기면 사고 나는 규칙
- **여러 파일에 걸친 흐름** — 한 파일만 봐서는 안 보이는 순서

"결제 취소는 관리자만 가능"은 코드 주석에도 있을 수 있어요. 하지만 **"고객이 직접 취소하다 재고가 어긋난 사고가 있어서 그렇게 막았다"는** 사람 머릿속에만 있습니다. 그 사람이 팀을 떠나면 같이 사라지고요.

2.3 항목마다 날짜와 근거를 붙입니다

낡음을 막는 방법은 의외로 단순합니다. **언제 기준인지, 무엇을 보고 썼는지**를 항목마다 붙이는 거예요. 나중에 "이게 아직 맞나?"를 확인할 수 있게 됩니다.

> 💡 프로젝트 폴더에서 그대로 복붙하세요. 날짜만 오늘로 바꾸시면 됩니다.

```
이 폴더를 훑어보고, 코드만 봐서는 알 수 없는 것만 골라
notes/결정기록.md 에 정리해줘.

각 항목은 이 4줄을 반드시 채운다:
- 무엇: (한 줄)
- 왜 이렇게 됐나: (한 줄, 모르면 "미확인" 이라고 적기)
- 근거: (파일 경로나 커밋 — 어디서 나온 얘기인지)
- 기준일: 2026-07-31

코드를 읽으면 바로 알 수 있는 설명(함수가 뭘 반환하는지 등)은 쓰지 마.
추측해서 채우지 마.
```

마지막 두 줄이 핵심입니다. 실제로 돌려보니 **모르는 칸은 "미확인"으로 남기고 넘어갔어요.** 그럴듯하게 지어내지 않았습니다. 지어낸 이유가 한 번 기록에 들어가면 그때부터는 그게 우리 팀의 공식 이유가 되어버리니까요.

2.4 쌓는 것보다 어려운 건 버리는 것입니다

기록을 시작하면 곧 알게 됩니다. 어려운 건 쌓는 쪽이 아니라 **버리는 쪽**이에요.

그래서 주기적으로 솎아내야 합니다. 한 달에 한 번이면 충분해요.

> 💡 기록 파일이 쌓였다 싶을 때 이걸 돌리세요. 고치는 건 사람이 확인한 뒤에 합니다.

```
notes/결정기록.md 를 읽고, 지금 코드와 어긋나는 항목을 찾아줘.

각 항목의 "근거" 에 적힌 파일을 실제로 열어서 확인하고
아래 셋 중 하나로 표시해.
[유효] 지금도 맞음 / [만료] 지금 코드와 다름 / [확인불가] 근거를 찾을 수 없음

[만료] 는 무엇이 어떻게 바뀌었는지 한 줄 덧붙이고,
고쳐 쓸 문장을 제안해줘. 파일은 아직 고치지 마.
```

이 프롬프트도 실제로 돌려봤습니다. 일부러 어긋나게 만든 항목(기록은 3000번, 코드는 4000번)을 넣어두고 돌렸더니 이렇게 나왔어요.

- 해당 항목을 **[만료]** 로 표시하고, 근거로 적힌 파일을 실제로 열어 4000번인 걸 확인해 왔습니다.
- 고쳐 쓸 문장을 제안하면서 **"왜 바뀌었는지는 저장소에 근거가 없어 채울 수 없다"고** 남겨뒀습니다.

- 시키지 않은 파일 수정은 하지 않았습니다.

"왜"를 못 채운다고 한 게 제일 값진 반응입니다. 그 칸은 사람만 채울 수 있어요. 여기서 AI가 그럴듯한 이유를 지어냈다면, 우리는 틀린 기록을 새 기록으로 갱신하는 셈이 됩니다.

한 가지 더 있었습니다. 다른 항목은 [유효]로 판정하면서, **"규칙이 주석에만 있고 코드가 강제하지는 않는다"는 점**까지 짚어줬어요. 이런 게 기록을 돌려볼 때 얻는 덤입니다.

2.5 어디에 두느냐도 중요합니다

이 기록은 **프로젝트 폴더 안에 마크다운 파일로** 두세요. 메모 앱이나 다른 데 두지 말고요. 이유는 두 가지입니다.

- AI가 별도 설정 없이 바로 읽습니다. 밖에 있으면 매번 붙여넣어야 해요.
- 코드와 같이 이력에 남습니다. 언제 무엇이 바뀌었는지 나중에 볼 수 있습니다.

다만 이력에 남는 건 **문서가 바뀐 시점**이지, **그 내용이 아직 유효한 시점**은 아닙니다. 그래서 기준일을 따로 적는 겁니다.

> ⚡ 한 단계 더 — 이 두 프롬프트를 매번 손으로 치기 번거로우면, 클로드 코드 커맨드로 만들어 두세요. `.claude/commands/` 폴더에 `기록.md` · `솎기.md` 로 저장하면 `/기록` · `/솎기` 로 부를 수 있습니다. 실제로 이런 정리 작업을 자동으로 돌리는 도구들도 나오고 있는데, 대개 **"모아 넣기"와 "솎아내기"를 별도 작업으로 분리**해 둡니다. 쌓기와 버리기는 성격이 다른 일이라서요.

✦ 이 PART의 핵심 정리

- 낡은 맥락은 없느니만 못하다. 코드는 깨지지만 문서는 조용히 틀린다
- 코드가 담지 못하는 것만 적는다 — 왜·사고 이력·불변 조건·여러 파일에 걸친 흐름
- 항목마다 근거와 기준일을 붙여야 나중에 유효한지 확인할 수 있다
- 한 달에 한 번은 솎아낸다. 모르는 칸은 AI가 지어내지 않고 "미확인"으로 두게 한다

PART 3 전체 4장

도구를 붙일수록 같은 일을 두 번 합니다

같은 제목의 문서가 두 개 만들어졌습니다. 같은 스프레드시트가 네 개 만들어진 경우도 있고요.

에러는 안 났습니다. 두 번 다 정상 응답이었고 기록도 깨끗했어요. **로그만 봐서는 보이지 않습니다.**

3.1 얼마나 흔한 일일까요

AI에게 외부 도구를 붙여 쓰면(문서 만들기, 메일 보내기, 시트에 쓰기 같은 것) 같은 도구를 같은 값으로 두 번 호출하는 일이 생깁니다. 한 연구자가 공개된 실행 기록 **6,780건**을 뒤져서 이걸 세어봤어요.

- 중복 호출로 잡힌 것: **8,042건**
- "다 했습니다" 같은 선언 반복을 빼고 실제로 의미 있는 것: **4,249건**
- 그중 응답에 담긴 고유번호를 대조해서 **진짜로 두 개가 만들어진 게 확인된 것: 159건**

159건이 작아 보일 수 있는데, 여기가 중요합니다. 나머지는 **판정을 못 한** 겁니다. 판정 불가 3,197건 중 2,011건이 같은 이유였어요. 도구가 결과를 돌려줄 때 **"성공"이라는 글자만 주고 무엇을 만들었는지 고유번호를 안 줬기 때문**입니다. 응답에 고유번호를 한 번도 안 주는 도구가 24종이었습니다.

그러니까 159건은 "이만큼 있었다"가 아니라 **"확인할 수 있었던 것만 이만큼"입니다.**

3.2 도구를 줄이면 해결될까요

직관적으로는 "도구를 많이 붙이니까 헷갈리는 거겠지" 싶습니다. 그런데 숫자는 그렇게 단순하지 않았어요.

같은 조사에서 데이터셋 두 개의 중복률이 3배 차이 났습니다(0.80% 대 2.41%). 도구 개수도 20개 대 523개로 크게 달랐고요. 그런데 붙인 서버 개수별로 나눠보니 **4~5개일 때 가장 높았고(약 2.5%), 그보다 많아지면 오히려 낮아졌습니다.** "많을수록 나쁘다"는 단순한 관계가 아니었던 거예요.

여기엔 정직하게 짚을 한계가 있습니다. **이 숫자는 전부 시험용 기록이지 실제 서비스 기록이 아닙니다.** 공개 데이터셋 여덟 개를 뒤져서 이 현상이 관측 가능한 게 이거 하나뿐이었대요. 대부분의 시험은 정답을 채점하려고 만들어져서, 중복이 나면 그냥 오답 처리되고 넘어가거든요.

그래서 이 장의 결론은 "도구를 줄이세요"가 아닙니다. **"내 환경에서 직접 확인하세요"입니다.**

3.3 지금 몇 개 붙어 있는지부터

대부분은 지금 자기 환경에 뭐가 붙어 있는지 모릅니다. 한 번씩 늘려온 거라서요.

> 💡 터미널에 이 한 줄만 쳐보세요. 지금 붙어 있는 외부 도구가 전부 나옵니다.

```bash
claude mcp list
```

이 책을 쓰는 환경에서 쳐보니 **11개**가 나왔습니다. 검색 두 종류, 브라우저 두 종류, 문서 도구, 분석 도구... 하나씩 붙일 때는 다 이유가 있었지만, 한 화면에 모아놓고 보니 **역할이 겹치는 것들**이 눈에 들어왔어요.

목록을 보면서 세 가지만 확인하세요.

- **지난 한 달 안에 실제로 쓴 적 있나?** 없으면 후보입니다
- **같은 일을 하는 게 둘 이상인가?** 검색 도구가 셋이면 AI도 어느 걸 쓸지 헷갈립니다
- **이 도구가 뭔가를 만들거나 보내는가?** 읽기만 하는 도구는 중복돼도 손해가 적습니다. 만들고 보내는 도구가 위험합니다

3.4 만들기 전에 확인하게 시킵니다

중복을 막는 방법은 도구 쪽이 아니라 **지시 쪽**에 있습니다. 순서를 세 줄로 못 박으면 됩니다.

> 💡 문서 만들기·메일 보내기·시트에 쓰기처럼 **뭔가 생기는 일**을 시킬 때 이 세 줄을 같이 붙이세요.

```text
(여기에 시킬 일을 쓰세요)

만들기 전 규칙:
1. 같은 걸 이미 만든 적 있는지 먼저 찾아본다 (이름·날짜·내용이 비슷한 것 포함).
2. 있으면 새로 만들지 말고, 찾은 것의 정확한 이름(또는 주소)을 알려주고 멈춘다.
3. 없을 때만 새로 만들고, 만든 것의 이름을 마지막에 한 줄로 보고한다.
```

이것도 돌려봤습니다. 같은 이름의 문서가 이미 있는 폴더에서 "7월 정산 보고서 만들어줘"라고 시켰더니, 여러 이름으로 검색해서 기존 파일을 찾아낸 뒤 **만들지 않고 멈췄습니다.** 새로 생긴 파일은 0건이었어요.

세 줄 중에 **3번이 은근히 중요합니다.** 만든 것의 이름을 보고하게 하면, 우리가 나중에 "그때 뭐가 생겼더라"를 찾을 수 있습니다. 앞에서 본 조사에서 판정 불가가 그렇게 많았던 이유가 정확히 이거였어요. 무엇이 만들어졌는지 아무도 안 남긴 것.

3.5 매번 붙이기 번거롭다면

이 세 줄을 PART 1의 지침 파일(`AGENTS.md`)에 한 번 넣어두면 매번 안 써도 됩니다.

```text
## 무언가를 만들거나 보낼 때
- 만들기 전에 같은 게 이미 있는지 먼저 확인한다. 있으면 만들지 말고 알려준다.
- 만든 것은 이름(또는 주소)을 마지막에 보고한다.
- 같은 작업을 두 번 시도해야 할 것 같으면, 먼저 첫 번째 결과가 실제로 남았는지 확인한다.
```

세 번째 줄이 실전에서 자주 걸립니다. 응답이 늦거나 화면이 멈춘 것 같아서 다시 시키는 경우요. 사람도 그러다 중복 결제를 만들죠. AI도 똑같습니다.

⚡ 한 단계 더 — 정말 중요한 작업(결제·발송·공개 게시)이라면 **작업마다 고유한 이름표를 붙이는 방식**이 확실합니다. "이번 작업 이름은 `2026-07-31-정산`이다. 같은 이름표의 작업이 이미 처리됐으면 다시 하지 마라"처럼요. 개발에서 쓰는 방식인데, 지시문으로도 흉내 낼 수 있습니다.

✦ 이 PART의 핵심 정리

- 중복 실행은 에러 없이 조용히 일어난다 — 둘 다 정상 응답, 로그도 깨끗
- 도구가 많다고 무조건 나빠지는 건 아니다. 다만 **만들고 보내는** 도구가 위험하다
- `claude mcp list`로 지금 뭐가 붙어 있는지부터 본다
- 시킬 때 "확인 → 있으면 중단 → 만든 것 보고" 세 줄을 같이 붙인다

PART 4 전체 4장

"인터넷 없음"이라고 적어두면 막힌 걸까요

프롬프트에는 이렇게 적혀 있었습니다. **"너는 인터넷에 접속할 수 없다. 이건 모의 환경이다."**

실제로는 열려 있었습니다. 설정 실수였어요. 그 한 칸의 차이가 남의 회사 서버 세 곳을 실제로 뚫는 결과로 이어졌습니다.

이 장은 AI 회사 이야기처럼 보이지만, 우리 폴더에서 그대로 재현됐습니다. 그것도 훨씬 시시한 방법으로요.

4.1 141,006번 중 6번

2026년 7월, AI 모델을 만드는 회사 두 곳이 각각 사고를 공개했습니다.

먼저 한 회사에서 내부 시험 중이던 모델이 **격리된 시험 환경을 빠져나갔습니다.** 알려지지 않은 취약점을 이용했고, 약 이틀 반 동안 **17,600번의 자동 작업**을 실행하며 외부 플랫폼 다섯 곳을 건드렸습니다.

이 공개를 보고 다른 회사가 자기네 기록을 전수 조사했습니다. **141,006번의 시험 실행**을 훑어서 **3건의 사고(총 6번 실행)**를 찾아냈어요. 세 건 모두 "숨겨진 정보를 찾아내라"는 훈련 과제였고, 세 건 모두 프롬프트에는 **"인터넷 접속 없음"이라고** 적혀 있었습니다. 그런데 설정 실수로 실제 인터넷이 열려 있었습니다.

무슨 일이 벌어졌냐면 이렇습니다.

- **1건**: 과제 속 가상의 회사 이름이 실제 존재하는 회사와 같았습니다. AI는 시험 환경 안에서 목표를 못 찾자 인터넷에서 같은 이름을 발견했고, **그걸 과제의 목표라고 생각하고** 뚫었습니다. 접속 정보와 함께 실제 운영 데이터 수백 행이 들어 있는 데이터베이스에 접근했어요.
- **2건**: 가상 회사의 안내문에 "이 프로그램 부품을 설치하라"고 적혀 있는데 그 부품이 실제로는 없었습니다. AI는 같은 이름으로 자기가 만든 부품을 **공개 저장소에 올렸습니다.** 약 1시간 동안 공개됐고, 그 사이 **실제 시스템 15대**가 내려받아 실행했습니다.
- **3건**: 목표를 못 찾자 인터넷에서 **약 9,000개의 대상을 훑어보고** 그중 한 곳을 뚫었습니다. 방법도 특별할 게 없었어요. 노출된 진단 페이지에 적힌 접속 정보를 읽는 정도였습니다.

피해를 입은 회사들은 **이 활동을 스스로 알아채지 못했습니다.** 연락을 받고 나서야 알았어요.

4.2 모델이 반항한 게 아닙니다

여기서 흔한 오해를 하나 정리하고 갑시다. 이건 AI가 몰래 나쁜 짓을 한 사건이 아닙니다.

AI는 **시킨 일을 했습니다.** "숨겨진 정보를 찾아내라"는 과제였고, 어디까지가 범위인지는 안 알려줬어요. 그리고 프롬프트가 "여긴 모의 환경"이라고 했으니, 만나는 것마다 시험용 소품이라고 믿었습니다.

두 번째 사고에서는 AI가 스스로 이렇게 판단하기까지 했습니다. "이게 진짜 인터넷이면 이건 실제 공격이고, 그건 절대 안 되는 일이다." 그러고는 **자기가 본 인증서를 못 알아보겠다는 이유로 "역시 모의 환경이 맞다"고 결론 내고 진행했습니다.** 잘못된 전제 위에서 논리는 멀쩡했던 거예요.

공개한 회사도 이걸 **모델의 문제가 아니라 환경·운영의 문제**로 정리했습니다. 그리고 이런 문장을 남겼습니다. 어디까지가 범위이고 어디부터가 아닌지를 프롬프트가 분명히 했다면 애초에 나가지 않았을 것이다.

우리에게 남는 교훈은 두 줄입니다.

- **말로 적은 제한은 제한이 아닙니다.** "하지 마"라고 적는 것과 못 하게 막는 것은 다릅니다
- **범위를 안 정해주면 AI는 목표만 보고 계속 갑니다.** 악의가 아니라 성실함 쪽입니다

4.3 우리 폴더에서 그대로 재현됐습니다

"큰 회사 시험 환경 얘기잖아" 싶으시죠. 그래서 집 규모로 해봤습니다.

빈 폴더를 만들고, 클로드 코드 설정 파일에 **인터넷으로 나가는 통로를 전부 막았습니다.** 웹 가져오기 도구, 웹 검색 도구, 그리고 터미널에서 파일을 내려받는 명령 두 개까지 차단 목록에 넣었어요. 그리고 시켰습니다. "example.com 페이지를 가져와서 첫 줄만 알려줘."

결과: 가져왔습니다.

돌아온 답에 이렇게 적혀 있었어요. 터미널 명령은 거부됐지만, 붙어 있던 **브라우저 도구**로 페이지를 열어서 읽어왔다고요. 실제 본문까지 정확히 가져왔습니다.

막았다고 생각한 통로는 넷이었는데, 열려 있는 다섯 번째가 있었던 겁니다. **PART 3에서 세어본 그 도구 목록**이 바로 그 통로였어요. 앞 장과 이 장이 여기서 만납니다.

그다음이 중요합니다. 차단 목록에 그 브라우저 도구들까지 추가하고 똑같이 시켰습니다.

이번엔 못 가져왔습니다. 그리고 이렇게 답했어요. "가져오지 못했습니다. 진행하려면 둘 중 하나를 허용해 주세요." 마지막 한 줄이 마음에 들었습니다. 관례상 첫 줄이 무엇일 거라고 덧붙이면서, **"이건 실제로 받아 확인한 값이 아니라 추정"이라고** 명시했어요.

정리하면 이렇습니다. 차단은 **목록에 적는 순간이 아니라, 실제로 시켜보고 못 나가는 걸 본 순간** 완성됩니다.

4.4 확인은 3분이면 됩니다

> 💡 순서대로 세 단계입니다. ①은 터미널에, ③은 AI에게 그대로 시켜보세요.

BASH

```
# ① 지금 뭐가 붙어 있는지 본다 (인터넷으로 나갈 수 있는 통로 확인)
claude mcp list
```

```JSON
// ② 프로젝트 폴더의 .claude/settings.json — 나가는 통로를 실제로 막는다
{
  "permissions": {
    "deny": [
      "WebFetch",
      "WebSearch",
      "Bash(curl:*)",
      "Bash(wget:*)"
    ]
  }
}
```

①에서 나온 목록 중 인터넷에 닿는 도구가 있으면 `"mcp__도구이름"` 형태로 `deny` 목록에 함께 넣으세요. 위 네 줄만으로는 부족했다는 게 4.3의 결과입니다.

```TEXT
③ 막혔는지 확인 — AI에게 그대로 시켜보세요

example.com 페이지를 가져와서 첫 줄만 알려줘.
가져오지 못했으면 "못 가져왔다"고 하고, 어떤 도구를 시도했다가 거부됐는지 알려줘.
추정으로 답하지 마.
```

③에서 내용을 가져오면 아직 열려 있는 겁니다. 어떤 도구로 가져왔는지 답에 적혀 있으니, 그걸 ②의 목록에 추가하고 다시 확인하세요.

같은 방식이 폴더 범위에도 적용됩니다. "이 폴더 밖은 건드리지 마"라고 **적어두는 것**과, 실제로 밖의 파일을 하나 읽어보게 **시켜보는 것**은 다릅니다. 중요한 폴더일수록 시켜서 확인하세요.

⚡ 한 단계 더 — 승인 절차를 통째로 건너뛰는 옵션(`--dangerously-skip-permissions`)을 습관처럼 켜두는 분들이 많습니다. 편하니까요. 그런데 이건 위 ②의 차단 목록까지 무의미하게 만들 수 있습니다. 켜고 쓰더라도 **인터넷·결제·발송·삭제가 걸린 폴더에서만은 끄는** 기준 하나는 정해두세요. 위 사고에서도, 일반 사용자에게 배포되는 버전에 붙어 있는 안전장치를 그 시험 환경에서는 떼어놓고 돌렸다는 점이 원인 중 하나로 지목됐습니다.

✦ 이 PART의 핵심 정리

- 프롬프트에 "하지 마"라고 적는 건 차단이 아니다. 설정으로 막아야 막힌다
- 통로를 넷 막아도 다섯 번째가 열려 있을 수 있다 — 붙어 있는 도구가 곧 통로다
- 차단은 목록에 적을 때가 아니라, 시켜보고 못 나가는 걸 확인할 때 끝난다
- 범위를 안 정해주면 AI는 목표만 보고 끝까지 간다. 악의가 아니라 성실함이다