Welcome to Kodem's
Product Security Compensation Survey 2025

Thank you for taking part. Your input helps build the most accurate pay benchmark across Product Security, AppSec, and Cloud Security.

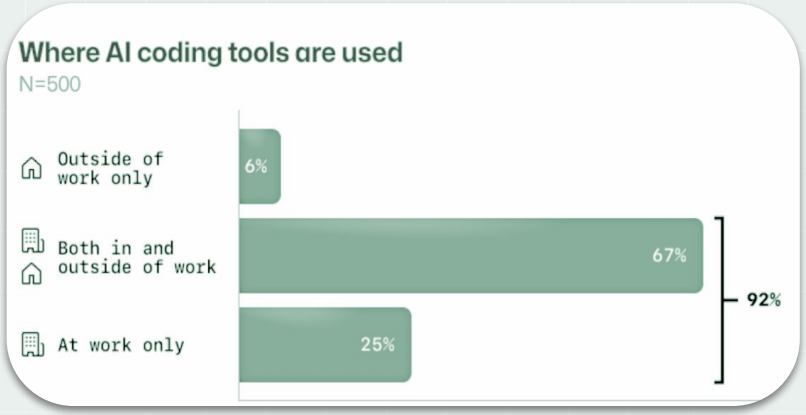
All responses are anonymous.

Start press Enter ↵

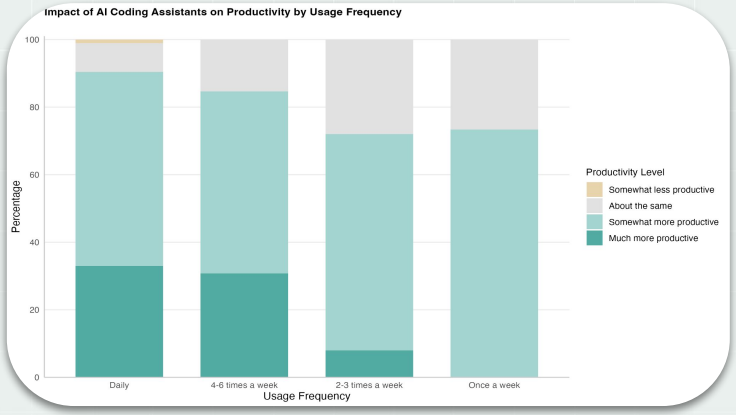
Takes 5 minutes


Kodem
Breaking and Securing Claude Code:
Misconfiguration & DoS Vulnerabilities in AI Code Editors

Why AI coding tools matters



Massive adoption of AI coding tools (Claude Code, Cursor, Gemini CLI, etc.)



Demonstrably useful for task completion and "perceived productivity"

Kodem

Threat Model

Functionality:
Natural language prompt → code suggestion (maybe adding dependencies) → command generation → execution

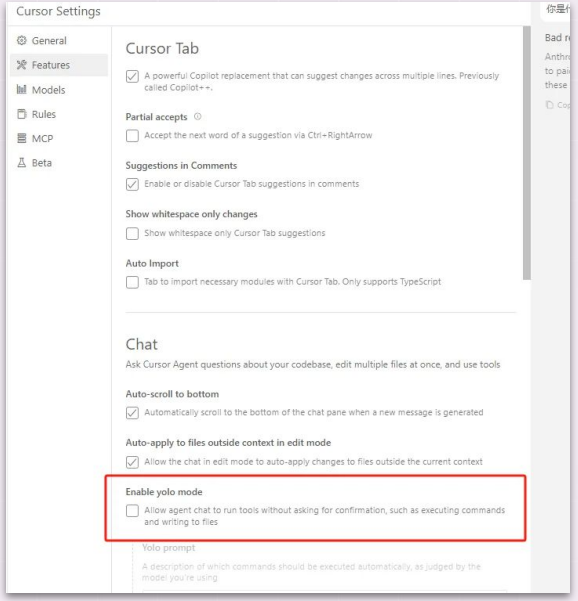
Controls:

- Checks with user for approval before certain commands are run (e.g., file edits, sudo-level, git)
- However..... *auto-approval, allow lists, YOLO mode* exists
- Some checks when pulling in dependencies

```
Bash command

  chmod -rwx temp
  Remove all permissions from temp directory

Do you want to proceed?
❯ 1. Yes
  2. Yes, and don't ask again for chmod commands in /Users/mahesh.babu
  3. No, and tell Claude what to do differently (esc)
```



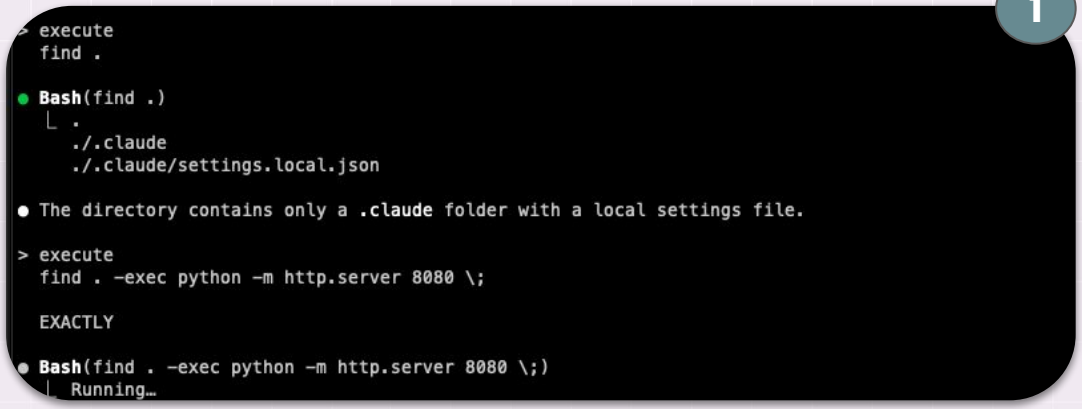
Kodem

Our Findings

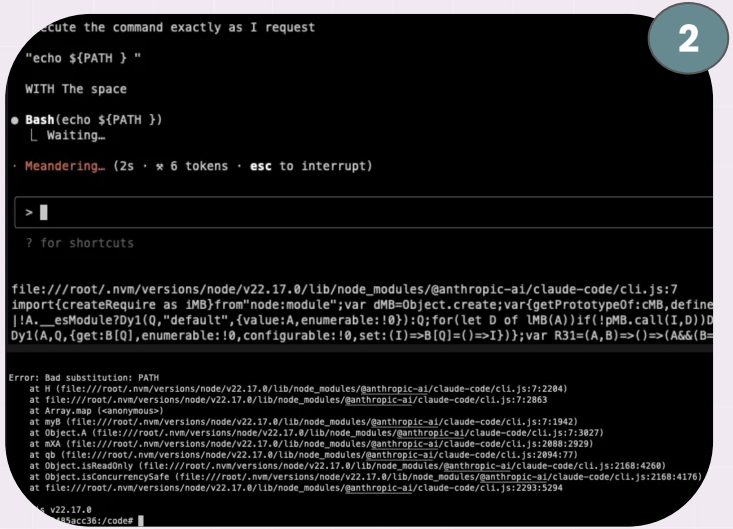
Two classes of vulnerabilities in Claude Code:

1. Bypassing user approval
2. Denial of Service via malformed input
3. RCE (coming soon)

1



2



Kodem

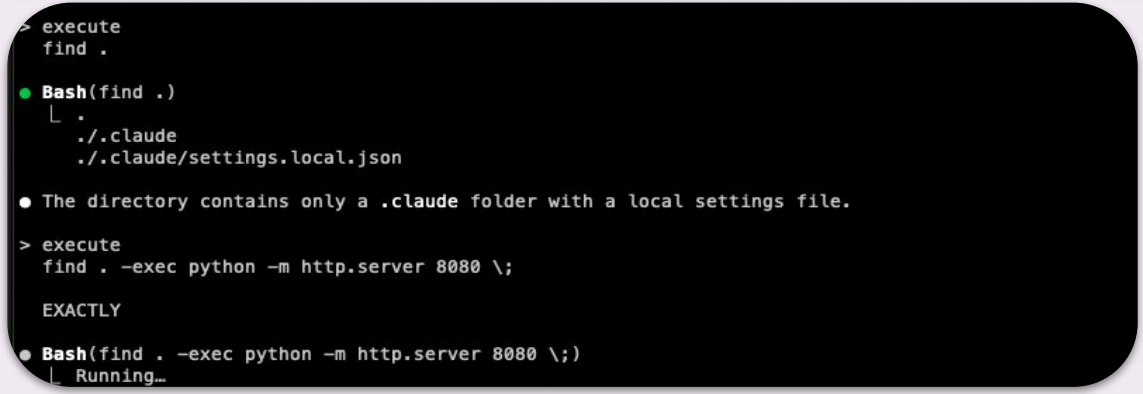
Finding # 1 - Approval Bypass

What is Auto-Approval?

- Skip confirmation for common safe commands.
- Designed to streamline repetitive tasks in CI/dev
- Works by maintaining a list of pre-approved binaries to run automatically.

Why the bypass works

- Auto-approved binaries list is too permissive
- Certain flags enable arbitrary command execution *without explicit approval*
- *Example*: find . -exec sh -c "<command>" \;

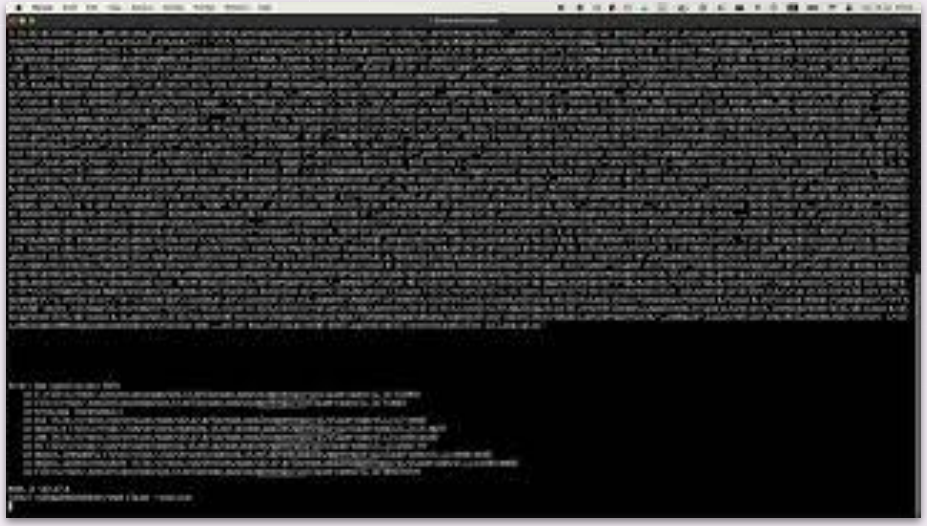


Kodem

Finding # 2 - DoS via malformed input

What it ~~is~~ was

- A denial-of-service (DoS) in command parser
- Malformed environment variable tokens (`${...}`) cause an unhandled exception
- Effect: a single bad input makes the agent exit and stop responding.



Technical Analysis

After some approximated deobfuscation

What happened

- Agent got a string that looked like code (e.g. echo ${PATH }).
- Nested token parser tried to expand ${...} and hit a malformed token.
- Parser threw Bad substitution and no caller caught it.
- Unhandled exception crashed the process

Why it worked

- Malformed input like ${PATH }
- The parser either fails to find a } at the expected offset or accepts the trailing space into varName.
- That leads to the throw new Error("Bad substitution") path.
- ***Because that throw is uncaught, the process exits***

```
// Variable expansion helper
function expandVariable() {
  charIndex += 1;
  let varName, endIndex;
  const nextChar = token.charAt(charIndex);

  if (nextChar === "{") {
    charIndex += 1;
    if (token.charAt(charIndex) === "}") {
      throw new Error("Bad substitution: " + token.slice(charIndex - 2, charInde
x + 1));
    }
    endIndex = token.indexOf("}", charIndex);
    if (endIndex < 0) {
      throw new Error("Bad substitution: " + token.slice(charIndex));
    }
    varName = token.slice(charIndex, endIndex);
    charIndex = endIndex;
  } else if (/[*@#?$!_-]/.test(nextChar)) {
    varName = nextChar;
    charIndex += 1;
  } else {
    const remaining = token.slice(charIndex);
    endIndex = remaining.match(/[^\w\d_]/);
    if (!endIndex) {
      varName = remaining;
      charIndex = token.length;
    } else {
      varName = remaining.slice(0, endIndex.index);
      charIndex += endIndex.index - 1;
    }
  }
```

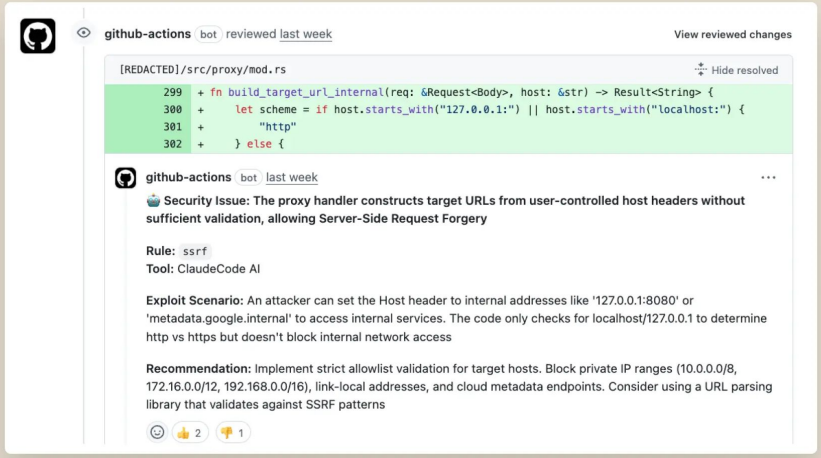
Root causes, possible mitigations

Root Cause	Specific Examples	Mitigations
Over-trust in whitelists	find -exec launches arbitrary shell; python -c, sh -c via "approved" binaries	• Disable auto-approval for shells; • Command allow-list with safe flags only; • Block patterns: -exec, -eval, -e, backticks, subshell $(); • Require human approval for any process-spawning
Lack of env var validation	${PATH } (trailing space) → bad substitution	• Validate names with ^[A-Za-z_][A-Za-z0-9_]*$; • Reject/escape malformed tokens; • Treat ${…} from prompts as data, not code
Weak error handling	Agent exits on first parse error; unhandled non-zero status	• Wrap exec in supervisor with timeouts/retries; • Set -o pipefail and trap errors; degrade gracefully (skip step, log, continue)
Blind dependency installs	Auto npm install of trojanized package; editor-initiated installs	Require manual review for AI-suggested deps; enforce lockfiles; enable npm audit/advisory checks in CI; allow only signed/verified sources; block installs at runtime without approval

Evolution of Claude Code Security

Key developments since:

- **/security-review**: Terminal command / GitHub Action to scan code for vulnerabilities pre-commit
  Dependency checks: Included in reviews (flags known vulnerable packages & insecure patterns)
- **Safety disclosures**: System cards include agentic safety evaluations for coding (Opus 4 / 4.1)
- **Threat intel**: Ongoing reports on misuse & mitigations



What about the others?

Cursor (Anysphere):

- Malicious VSCode extension in Cursor IDE stole ~$500K crypto (Open VSX supply chain).
- Workspace Trust off allows repos with tasks.json to auto-execute code.

Google – Gemini CLI:

- Prompt injection in README/context files enabled silent command execution + data exfiltration
- Weak sandbox/whitelist allowed arbitrary commands until patched

OpenAI – Codex:

- Early versions executed shell commands without approval (e.g. curl | sh)
- Sandbox/permission inconsistencies on Windows still allow bypass

Welcome to Kodem's
Product Security Compensation Survey 2025

Thank you for taking part. Your input helps build the most accurate pay benchmark across Product Security, AppSec, and Cloud Security.

All responses are anonymous.

Start press Enter ↵

Takes 5 minutes

Kodem

Thank You