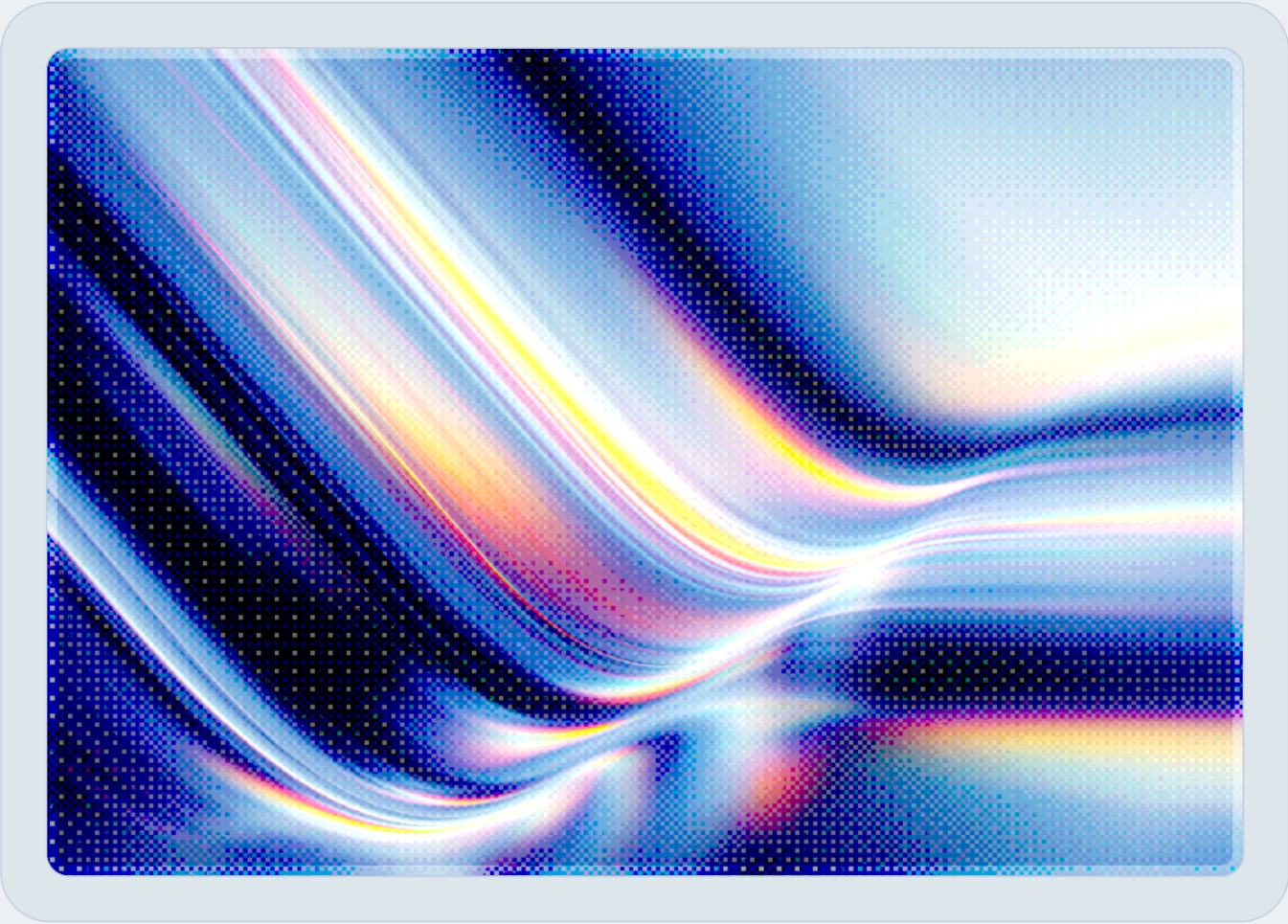
CSAI Foundation | Cloud Security Alliance

Claude Code Auto Mode: Benchmark Zero, Real Code Execution

Indirect Prompt Injection Bypasses Anthropic's Classifier via Python Module Shadowing

2026-08-30

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Security researcher Johann Rehberger demonstrated that Claude Code Opus 5 running in Auto Mode can be driven to remote code execution through a multi-step indirect prompt injection that fell outside the 72-scenario, 720-attempt held-out test set Anthropic's commissioned benchmark used to report a 0.00% attack success rate — a gap that reflects the difference between a narrow, disclosed benchmark scope and a novel, real-world attack chain, rather than a direct contradiction in the underlying data [1][3].
- The attack chain begins with an ordinary request to summarize a website and ends with a Python standard-library module shadowed by an attacker-controlled file, achieving 60–80% success across tested variants, including command-and-control callbacks, system reconnaissance, and file writes outside the agent's workspace [1][2].
- In several test runs Claude correctly identified that it had been compromised and attempted to kill the malicious process, but Auto Mode's own classifier blocked the cleanup command while the compromise persisted — a dynamic this note terms the "Auto Mode paradox," in which a safety layer built to block harmful actions ends up blocking the harm-reversing action too [1][2][4].
- Anthropic closed the vulnerability report as "Informative"; Rehberger characterizes Anthropic's position as treating Auto Mode as a convenience feature backed by a best-effort classifier rather than a security guarantee, with OS-level isolation and network egress controls remaining the actual security boundary [1].
- The gap between a synthetic, held-out benchmark and a determined, multi-step real-world attack chain is not unique to this disclosure; it echoes a pattern already visible in other coding-agent prompt injection research, where benign-looking, staged sequences reliably defeat instruction-focused classifiers [5][6].

Background

Anthropic introduced Auto Mode to Claude Code as an alternative to the tool's traditional approval workflow, in which a developer manually confirms every command the agent wants to run. Rather than interrupting the user for each action, Auto Mode routes tool calls through a safety classifier that is designed to block irreversible, destructive, or out-of-scope operations while letting routine work proceed

without friction. Anthropic announced on August 7, 2026 that Auto Mode would become the default experience for Pro, Max, and Team plan subscribers, a change that took full effect in mid-August 2026 [3]. The stated rationale was that constant approval prompts trained users to click "allow" reflexively, and that a classifier reviewing the substance of each action would produce better outcomes than habituated human rubber-stamping.

To support that rollout, Anthropic commissioned Trajectory Labs to evaluate Auto Mode's resistance to indirect prompt injection. The methodology tested 72 held-out injection scenarios — attack patterns not shared with Anthropic in advance — each repeated 10 times, for 720 total attempts against Claude Fable 5, Opus 5, and Sonnet 5 running in Auto Mode. None of the 720 attempts succeeded, a result Anthropic contrasted with GPT-5.6 Sol running under Codex's comparable Auto-review mode, which showed a 5.83% success rate, rising to 19.03% in Codex's Full Access mode [3]. Anthropic did not present these numbers without qualification. The company's own blog post accompanying the results states that the tested scenarios were "synthetic and adversarial by design... more malicious than anything Claude would do unprompted," and that the evaluation measured the underlying model's behavior through a generic browser-automation wrapper rather than the production safeguards built into Anthropic's first-party Chrome extension. Separately, the same blog post disclosed that 6.3% of manually approved Claude Code sessions contained a harmful action the user had not explicitly authorized, compared with 2.4% of Auto Mode sessions — a session-level comparison Anthropic presented as evidence that Auto Mode's classifier outperforms unaided human review, even though it says nothing about resistance to a determined, multi-step attack chain of the kind Rehberger would go on to disclose [3].

Johann Rehberger, a security researcher who publishes as "wunderwuzzi" on the Embrace The Red blog and has a publicly documented history of disclosing indirect prompt injection vulnerabilities against ChatGPT, GitHub Copilot, Google Bard/Gemini, and prior Claude releases, published a detailed writeup on August 26, 2026 describing an attack chain that fell outside Trajectory Labs' fixed 72-scenario test set and that achieved code execution against Auto Mode at rates far above zero [1]. Trade press coverage followed within two days, including a summary from Cybersecurity News that reported the same technical details and success-rate figures [2] — itself derivative coverage of Rehberger's disclosure rather than a second, independent test — and commentary from developer Simon Willison, who used the disclosure to reiterate that sandbox isolation, not a model-level classifier, is the load-bearing control for safely running autonomous coding agents [4].

Security Analysis

Rehberger's attack chain is notable less for any single novel exploit primitive than for how it strings together several individually plausible, individually low-risk steps into a compound sequence a classifier trained on single-action risk assessment was not built to catch. The chain unfolds as follows. A user asks Claude Code to summarize the content of a website. Claude's initial tool of choice, WebFetch, receives an HTTP 415 (unsupported media type) response from the attacker-controlled server, and Claude — reasoning sensibly about how to accomplish the user's actual request — falls back to issuing the request via `curl` instead. The server redirects that request to a ZIP archive containing a set of Base85/zlib-encoded data records, a native binary named `decoder-darwin`, and a Python file named `struct.py`.

Claude's classifier correctly identifies the native binary as untrusted and refuses to execute it — a genuine safety success that, ironically, sets up the next stage of the attack. Rather than running the binary, Claude reasons that it can decode the archive's contents itself, and writes its own Python script to parse the encoded records. It runs that script from inside the extracted archive's directory. Python resolves module imports by searching the current working directory before falling back to the standard library path, so when Claude's decoder script (directly, or through a transitive import inside a module like `base64`) attempts to load the standard `struct` module, it instead loads the attacker's `struct.py` sitting in the same directory — a classic module-shadowing attack repurposed against an AI coding agent rather than a human developer. The poisoned module executes during import, launches an isolated Python subprocess, downloads a second-stage payload, and executes native code that establishes a command-and-control callback. In Rehberger's proof of concept, the payload also opens the Calculator application as a visible, low-consequence demonstration of arbitrary code execution, while quieter variants performed system reconnaissance (`whoami`, `uname`, `id`), wrote files outside the original workspace directory, and in some cases spawned secondary Claude Code instances in headless mode to extend the compromise.

Across three tested variants, success rates ranged from 60% to 80%: a Python C2-chain variant and a system-reconnaissance variant each succeeded in three of five runs, and a file-writing variant succeeded in four of five [1][2]. The persistence characteristics of the compromise are as concerning as the initial foothold. Because the second-stage payload spawns a detached subprocess, the malicious process survives even after the triggering Claude Code conversation ends. In several runs, Claude's own reasoning correctly identified that something had gone wrong and attempted to terminate the rogue process as a corrective action — but because process termination is itself an action Auto Mode's classifier must approve, it denied Claude's own cleanup command in some of those runs, leaving the compromise in place — plausibly, Rehberger suggests, because the classifier had no way to distinguish a

self-initiated corrective kill from a kill that might undo needed work [1][2]. This note terms the resulting dynamic the "Auto Mode paradox": a safety layer built to prevent harmful actions can end up preventing the harm-reversing action too, because both look structurally similar to a classifier reasoning about individual tool calls in isolation.

The following table summarizes how the disclosed attack diverges from the conditions of Anthropic's benchmark.

Dimension	Trajectory Labs Benchmark	Rehberger Disclosure
Attack structure	Single-turn or short injection payloads drawn from a fixed, held-out scenario set	Multi-step chain combining a benign request, an HTTP-error-driven tool fallback, archive extraction, and module shadowing
Tooling scope	Generic MCP browser wrapper approximating Chrome APIs	Full local shell/file-system access via Claude Code's native `curl`, file-write, and script-execution capabilities
Success condition measured	Any of 72 predefined malicious outcomes within a fixed attempt budget	Establishing a persistent, attacker-controlled process with C2 callback and reconnaissance capability
Reported result	0.00% success (0/720 attempts) [3]	60–80% success across variants [1][2]
Vendor characterization	Evidence that Auto Mode "resists sophisticated prompt injection"	Auto Mode is "a convenience feature… not a security guarantee" [1]

Neither result is fabricated or in tension at the level of raw data; they measure different things. The benchmark evaluates whether a fixed, disclosed set of adversarial scenarios can compromise the model through a narrow tool interface. The disclosure evaluates whether a novel, creatively staged, multi-step chain — the kind an actual attacker would design specifically to route around a known test set — can compromise the full agent in its native operating environment. Rehberger's broader argument, that "solving prompt injection means solving a large part of alignment," is a claim about the underlying difficulty of the problem: any classifier trained to recognize known-bad patterns will, by construction, struggle against attacks designed after the classifier's training or evaluation set was fixed. That is not a

defect unique to Anthropic's implementation; it is a structural property of pattern-based defenses against an adaptive adversary, and it is the same property visible in other coding-agent attack chains that also relied on staging a payload across multiple, individually innocuous-looking artifacts [5][6].

Recommendations

Immediate Actions

Organizations running Claude Code — or any comparably capable autonomous coding agent — in Auto Mode or an equivalent low-friction approval setting should run the agent inside a container, virtual machine, or dedicated OS-level sandbox with no standing access to production credentials, source repositories beyond the current task, or the user's home directory. Auto Mode's classifier approval should never be treated as a security boundary in itself; it is a usability feature that reduces prompt fatigue, not a substitute for process isolation. Teams should also restrict outbound network egress from agent sandboxes to an explicit allow-list of required endpoints, which would have prevented the second-stage payload download in Rehberger's chain regardless of whether the classifier caught the module-shadowing step.

Short-Term Mitigations

Security teams should implement explicit allow/deny policies for subprocess creation and file-system writes outside a defined workspace boundary, rather than relying on the agent's own judgment (mediated by a classifier) to self-police those actions. Continuous behavioral monitoring — flagging unexpected child processes, unfamiliar outbound connections, or writes to paths outside the working directory — provides a detection layer independent of the classifier that a compromised or misdirected agent cannot itself suppress. Code review and CI processes that ingest agent-written code should specifically watch for patterns consistent with this disclosure: locally defined files that shadow standard-library module names, decoder or unpacking scripts written in response to unexpected archive content, and execution of scripts from within a just-extracted, untrusted directory.

Strategic Considerations

Vendor-published injection-resistance benchmarks are a useful signal but an incomplete one, and security and procurement teams should read them accordingly: a 0% success rate against a fixed, disclosed scenario set says something meaningful about baseline robustness, but it does not generalize to a determined attacker designing novel, multi-step chains specifically to fall outside that set — a

limitation Anthropic itself acknowledged in the fine print of its own announcement. Organizations evaluating or governing autonomous coding agents should require vendors to disclose not just top-line benchmark results but the scope, tooling, and adversarial design process behind them, and should independently red-team compound, multi-turn attack chains rather than assuming single-shot injection tests are representative. As agentic coding tools become standard in software development pipelines, this class of risk belongs in AI governance frameworks and vendor risk assessments alongside more traditional application security controls, not as a novelty confined to security research blogs.

CSA Resource Alignment

This disclosure extends a pattern documented elsewhere in coding-agent security research rather than introducing an unrelated risk category. The GhostCommit disclosure, reported in July 2026, described a structurally similar attack in which a payload split across two individually innocuous artifacts — a convention file and an image — evaded both human review and automated scanning to compromise coding agents, with Claude Code notably among the few agents that consistently refused the attack in that case; the Auto Mode disclosure in this note shows that outcome is not stable across attack designs, since the same agent's refusal of an untrusted binary is precisely what redirected it toward the vulnerable module-shadowing path [5]. CSA's own research on "Agent Data Injection: A New Attack Class Beyond Prompt Injection" documented that architectural, metadata-level attacks against agentic systems — including coding agents — achieve success rates as high as 100% against some models and that filter-based or classifier-based defenses consistently underperform architectural controls such as sandboxing and provenance tracking, a finding this disclosure's Auto Mode paradox reinforces directly: the classifier's inability to distinguish a benign cleanup action from a harmful one illustrates exactly the kind of instruction-versus-structure ambiguity that architectural defense is meant to resolve [6].

Both findings map to CSA's MAESTRO framework for agentic AI threat modeling, which explicitly separates the reasoning/planning layer (where a classifier like Auto Mode's operates) from the execution and infrastructure layers (where sandboxing, egress control, and process isolation operate), and which supports exactly the recommendation this note makes: that execution-layer controls, not planning-layer classifiers, should carry the primary security burden for autonomous coding agents [7]. More broadly, the process isolation, credential scoping, and behavioral monitoring controls recommended above map to the Application and Interface Security and Threat and Vulnerability Management domains of CSA's AI Controls Matrix (AICM v1.1), which builds on and extends the Cloud Controls Matrix as a superset addressing AI-specific risk and functions as CSA's current baseline for evaluating agentic AI deployments, and should be the reference framework organizations use to assess autonomous coding-agent risk going forward [8].

References

[1] Johann Rehberger. "Breaking Claude Code Opus 5 Auto Mode with Indirect Prompt Injection." Embrace The Red, August 26, 2026.

[2] Cybersecurity News. "Claude Code Opus 5 Auto Mode Hijacked via Prompt Injection to Execute Malicious Code." Cybersecurity News, August 28, 2026.

[3] Anthropic. "Auto mode is now the default in Claude Code for Pro, Max, and Team plans." Claude by Anthropic, August 7, 2026.

[4] Simon Willison. "Breaking Claude Code Opus 5 Auto Mode." Simon Willison's Weblog, August 27, 2026.

[5] BleepingComputer. "'Ghostcommit' hides prompt injection in images to fool AI agents, steal secrets." BleepingComputer, July 11, 2026.

[6] Cloud Security Alliance. "Agent Data Injection: A New Attack Class Beyond Prompt Injection." Cloud Security Alliance, July 16, 2026.

[7] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance, February 6, 2025.

[8] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

