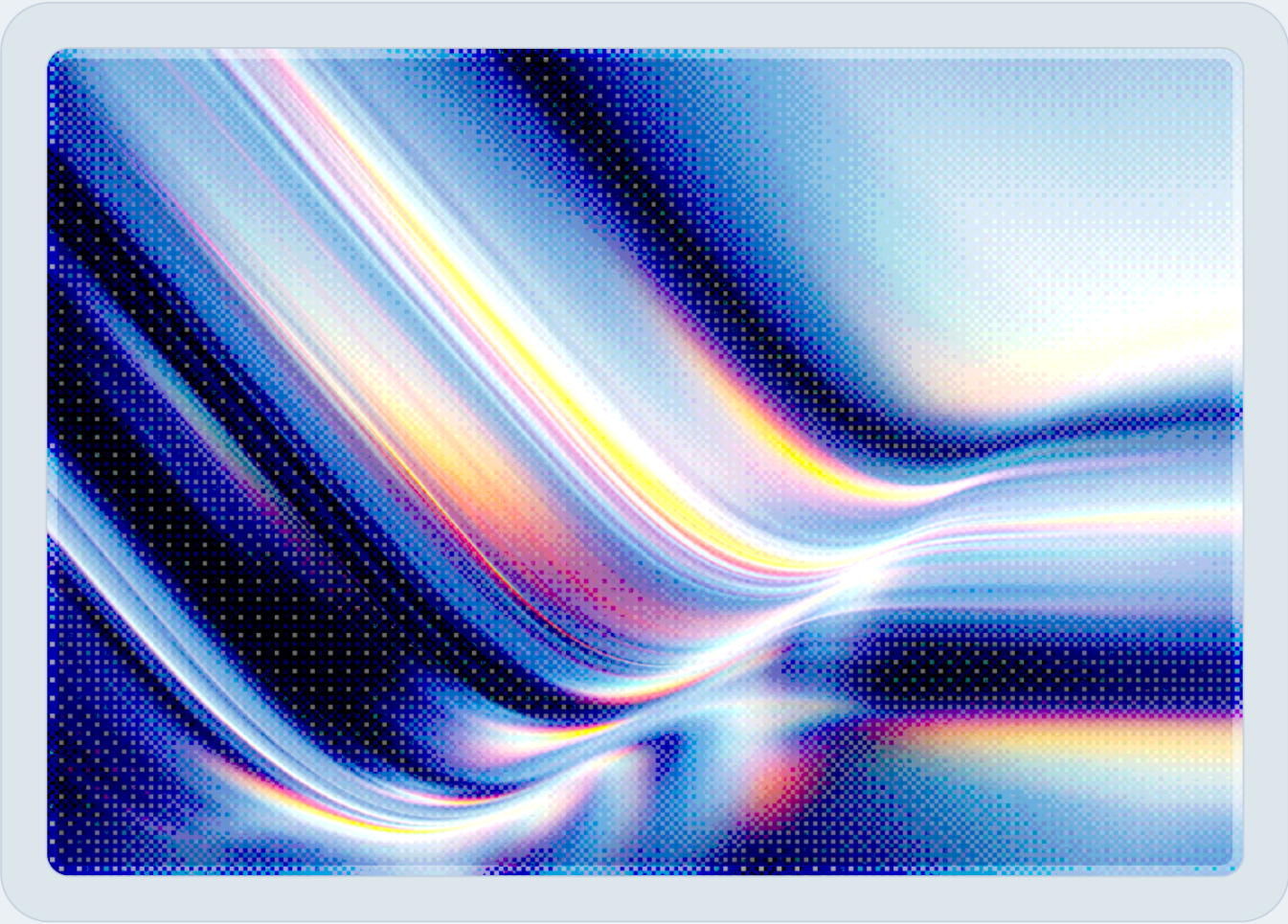
CSAI Foundation | Cloud Security Alliance

Poisoned at the Source: Search APIs Feeding Malicious Content to Coding Agents

Security Implications and Guidance

2026-11-24

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

AIR Research Lab released the WebSec benchmark on November 23, 2026. AIR describes it as the first public measurement of how often commercial web search APIs give AI coding agents malicious or risky content during ordinary development work [1][2]. The researchers ran 4,719 realistic coding-agent queries against eight search providers and scanned more than 282,000 results. None of the queries mentioned anything malicious, yet every provider returned known-malicious content at least once [1][2]. Across the full set of searches, agents were shown malicious or risky content 1,026 times. Counting only known-malicious content, 534 of roughly 37,700 provider searches (about 1.4%) were affected, and individual providers ranged from 0.8% to 2.2% [1][2][3]. The benchmark was produced by a security vendor and has not been independently replicated, but its methodology is published in detail.

The findings move the trust boundary for coding agents upstream. Earlier attention went to poisoned repositories, MCP servers, and package registries. The WebSec data shows that the retrieval layer itself, meaning the search API an agent calls to debug an error or choose a library, serves content across several threat categories, including attacker-controlled instructions, lookalike domains, invented package names, and download-and-execute commands. A documented example shows the consequence. A GitHub repository with 286 stars ranked as the top result for a Solana starter-kit query, and its only install step was a PowerShell one-liner that deployed a NetSupport Manager remote access tool [4].

For security teams, the first conclusion is that search results are untrusted input and should be handled with the same suspicion as user-supplied data, not as vetted reference material [1][3]. Exposure also scales with usage. AIR estimates that at a 1% rate, an organization whose agents run 5,000 searches a day would put known-malicious content in front of an agent about 50 times a day [3]. Provider choice matters but is not a control in itself: on AIR's WebSec Index, the gap between the safest and riskiest provider was more than fourfold, yet no provider came back clean [1]. Finally, the highest-risk moment is installation. Exposure rates were highest for install and package-selection queries, so download-and-execute steps need human review or policy enforcement [3].

Background

Many coding agents now rely on live web search to stay useful. A model's training data cannot cover a library released last month, an error message introduced in a recent framework version, or a deprecated API. Agent harnesses therefore give the model a search tool, which is usually backed by a commercial search API such as Brave, Exa, Firecrawl, Perplexity, Tavily, or You.com. These services return ranked results, and often page content, straight into the agent's context window. The agent then reasons over that content and acts on it: it edits files, runs shell commands, and installs dependencies. In common agent configurations, search results are passed to the model without any intermediate check for hostile content.

Concern about this pathway has grown over the past year. In April 2026, Google Threat Intelligence and Google DeepMind reported a scan of Common Crawl archives. It found a 32% relative increase in malicious indirect prompt injections on the public web between November 2025 and February 2026 [6]. Google described most of what it observed as low in sophistication. The examples were mainly pranks, SEO manipulation, and attempts to deter AI crawlers, and Google did not observe large-scale use of advanced exfiltration techniques. Even so, the company said it expects the threat to grow as attackers automate their own operations [6]. In parallel, academic work has shown that code-generating models routinely recommend packages that do not exist. A USENIX Security 2025 study of 16 models found average hallucinated-package rates of at least 5.2% for commercial models and 21.7% for open-source models, and recorded 205,474 unique hallucinated package names [7]. Attackers can register those names, a technique the community calls "slopsquatting."

AIR Research Lab's earlier work linked these trends to search APIs specifically. Its October 2026 "Web Hole" study ran about 2,000 developer queries through an agent's search tool and examined roughly 6,500 sites. It found 36 results pointing to domains that were no longer registered and could be bought by anyone. Examples included `pyquestions.com` and `thestackoverflow.com`, which surfaced for routine Python and AngularJS questions [5]. The researchers observed that "a site's rank in the search index can stay the same even after the site stops existing or its content changes." This means a buyer of an expired domain inherits the domain's search placement [5]. A few days later, the SolanaCry write-up documented a working malware delivery chain that relied on search ranking and an agent-readable README [4]. The WebSec benchmark extends those case studies into a systematic, multi-provider measurement.

Security Analysis

What the WebSec Benchmark Measured

The benchmark's method is designed to reflect what coding agents actually search for, not what an adversary might plant in a query. According to AIR's contributed article, the team started from 8,684 recorded sessions of open-source coding agents fixing real GitHub issues. It kept the 850 sessions with the most material to search for and used a model to generate 50 plausible developer searches for each, producing 42,296 candidate queries [2][3]. Two separate judge models scored the candidates for realism, and only queries in the top 30% for both were kept [2]. The article reports that 3,220 queries survived judging and that 1,499 package-selection and installation queries were then added, for a final set of 4,719 [3]. The benchmark site describes the same pipeline somewhat differently, characterizing the seed sessions as common prompts for different tasks and attributing the final count to deduplication with no more than five queries per task [2]. Every result was fetched twice, once with an AI-agent profile and once with a browser profile, to detect content shown only to agents. Each result was then run through a set of detectors, and the research team reviewed every finding [2][3]. The published materials do not name the models used to generate or judge the queries.

Scoring rests on a per-query severity measure. Each reviewed finding carries a detector weight, with confirmed malicious content weighted up to 100. A provider's Risk Index is the average severity it served per query, and the headline WebSec Index inverts and normalizes that figure so that higher scores mean safer results [2]. The published scores are summarized below. The benchmark site and blog say all eight providers were ranked, but the public text gives numeric index values for only three of them [1][2].

Provider	Published WebSec Index	Reported Position
Firecrawl	21.59	Highest (safest)
Tavily	18.59	Second
Perplexity	Not published in text	Among the top group
Parallel, Exa, Nimble	Not published in text	Not stated; implied mid-ranking
Brave	Not published in text	Among the lowest
You.com	5.02	Lowest

Two limits on these figures matter for practitioners. First, the published materials do not say which provider had the 0.8% exposure rate and which had the 2.2% rate, so readers cannot reconstruct per-provider exposure from public sources alone [2][3]. Second, AIR is a security vendor, and the analysis was published in The Hacker News as a contributed partner piece, not as peer-reviewed research [3]. The method is described in enough detail to be credible. Independent replication would still strengthen confidence in the relative rankings, which may also shift as providers change their indexes and filtering.

Six Pathways from Search Result to Compromise

The benchmark groups its detections into six threat categories. Each one exploits a different gap between how a search engine ranks content and how an agent consumes it. The categories and detectors below come from the benchmark site and the accompanying article [2][3].

Threat Category	Mechanism	Example Detectors
Prompt injection	Hidden text, HTML comments, or README instructions addressed to the agent; pages that show agents different content from browsers	Agent-versus-browser content differences
Typosquatting	Lookalike vendor domains, packages one character off a popular library, impersonated repositories	Lookalike domain; lookalike of known-malicious
Hallucination attacks	Registering package names or domains that models invent	Nonexistent package; registrable domain
SEO/AEO manipulation	Repositories and pages built to rank for agent-style queries, including inflated star counts	New unranked domain; low-reputation repo
Web search hijacking	Taking over a page that already ranks	Expired domain; subdomain takeover; re-registered domain; removed page
Manipulative content	Pages recommending malicious packages or unsafe installation patterns	Malicious package; download-and-run; `curl | bash`

These categories are especially dangerous for agents because agents do not reliably apply the contextual judgment a human developer might. A developer might notice that a repository is a week old, that a domain name is slightly misspelled, or that an install instruction pipes a remote script into a shell from an unfamiliar host. Agents appear prone to treating top-ranked results as authoritative, and instruction-following behavior can lead them to carry out the steps a README describes, as the SolanaCry case illustrates [4]. The benchmark reports that install and package-selection queries had the highest exposure rates, although no per-category figures were published [3]. Installation is the point where a poisoned result turns most directly into code execution.

Anatomy of a Real Chain: SolanaCry

The SolanaCry case shows how these pathways combine in practice. AIR found a GitHub repository, `SolanaUpdateTools/solana-starter-kit`, created in November 2025 and carrying 286 stars. It was the top search result for a Solana starter-kit query [4]. The README's only install step was a PowerShell one-liner that assembled the URL `developer-update.app/update/indicator`, downloaded it with `iwr`, and ran it in memory with `iex` [4]. The payload URL was no longer serving the file by the time of analysis, but the researchers recovered a roughly 21 MB dropper from the Web Archive. The dropper wrote genuine, signed NetSupport Manager components into a hidden directory alongside an attacker-supplied configuration file. That file pointed the client to a command-and-control server, hid the user interface, and created a Run-key persistence entry [4]. AIR linked a configuration value to a ClickFix cluster that eSentire had previously documented, which suggests the repository was part of an existing commodity malware operation and not a one-off proof of concept [4].

The chain needed no exploit against the agent, the model, or the search provider. It used inflated social proof to rank, a README written as instructions an agent would follow, and a download-and-execute pattern that agent harnesses with auto-approval enabled may execute without a prompt. AIR's write-ups state that Claude Code's web search runs on Brave Search, and they place Brave near the bottom of the WebSec ranking [1][4]. Readers should treat that provider attribution as AIR's claim and confirm current backend arrangements with their own tool vendors. The broader point does not depend on any single provider. Whichever index an agent queries, the content it returns runs with the agent's privileges.

Why This Differs from Prior Coding-Agent Threats

Earlier coding-agent attacks often required the victim to take a specific action first, such as cloning a poisoned repository, installing a compromised MCP server, or connecting the agent to an attacker-writable data source. Search-layer poisoning removes the need for any hostile action by the developer beyond ordinary use of a search-enabled agent. The developer only has to ask the agent to fix a common

error, and the agent pulls hostile content into its own context. Recent academic work suggests this class of attack can become more patient. The "Earn-then-Exploit" study, published on arXiv on October 31, 2026, describes fraudulent sites that first give a coding agent useful help to build trust and then steer it toward harmful actions outside the user's intended scope. The authors evaluated the technique across 56 organizational environments [8]. Combined with search ranking manipulation, such content could look legitimate in early interactions and weaponize later ones.

The attack also fits the economics attackers prefer. Expired domains with existing search placement can be registered by anyone [5]. Hallucinated package names are generated at scale, giving attackers a large pool of candidate names to register [7]. Repository stars can be inflated, and new domains can rank within days [3]. The OWASP Top 10 for LLM Applications already lists prompt injection, including indirect injection through external content, as the leading risk category [9]. The WebSec data adds a measured base rate for how often that external content reaches coding agents through mainstream retrieval infrastructure.

Recommendations

Immediate Actions

Organizations running coding agents with web search enabled should first remove automatic execution of download-and-execute patterns. Shell commands that pipe remote content into an interpreter, such as `curl | sh`, `curl | bash`, and PowerShell `iwr | iex`, should require explicit human approval no matter how the agent found them, and auto-approval for dependency installation from unverified sources should be turned off [3]. Such a control would have interrupted the agent-driven path of the SolanaCry chain at the execution step, provided the approver recognized the obfuscated PowerShell one-liner as a remote-execution pattern [4]. Security teams should also confirm which search backend each approved agent tool uses, because that choice now carries a measurable difference in exposure [1] [2].

Teams should then review recent agent activity for signs of exposure. AIR recommends logging what agents read, not only what they run [3]. Where agent transcripts or tool-call logs are kept, defenders should look for package installations that followed web searches, executions of remote scripts, and connections to recently registered domains. Any host that ran a `developer-update.app` one-liner or connects to the indicators published in the SolanaCry analysis should be investigated for NetSupport Manager persistence [4].

Short-Term Mitigations

Over the next several weeks, organizations should place a verification layer between agents and package managers. Before an install runs, the dependency should be confirmed to exist in the official registry, its age and maintainer history should be checked, and it should be screened against malware advisories such as the OSV database [3][10]. Packages that are newly published or that the registry does not recognize should block the install by default. Doing this through a package firewall or proxy, not through agent prompting, keeps the control in place even when the model is persuaded to bypass it. Domain reputation checks should follow the same reasoning, with domain age and registration history checked directly instead of assumed from search rank [3][5].

Search results themselves should be handled as untrusted data before they reach the model. Practical measures include removing hidden text and HTML comments from retrieved pages, flagging pages that serve different content to agent and browser user agents, and limiting search for sensitive repositories to domain allowlists of official documentation and registries. None of these measures fully solves prompt injection, and OWASP guidance recommends privilege restriction and human approval alongside content filtering [9]. Agents should run in isolated environments without standing access to production credentials, cloud tokens, or signing keys, so that a successful injection has limited reach.

Strategic Considerations

Over the longer term, search API selection should become part of AI supply chain governance. Procurement and architecture reviews for agent platforms should ask providers how they detect expired-domain takeovers, agent-targeted cloaking, and known-malicious repositories, and whether they publish safety metrics. The WebSec benchmark gives a starting point for that conversation, but organizations should push for independent, repeatable measurement and avoid relying on a single vendor-published ranking [1][2].

Security architects should also recognize that the retrieval layer is now part of the software supply chain. Controls developed for package registries, such as provenance, reputation, and allowlisting, need equivalents for the web content agents consume. Threat models for agentic development environments should treat the search tool as an untrusted input channel with the same severity as an untrusted MCP server, since both inject arbitrary third-party text into a privileged agent context. Detection engineering should correlate agent reads with agent actions, so that an investigation can trace a malicious install back to the search result that triggered it.

CSA Resource Alignment

This note extends a line of CSA AI Safety Initiative research that treats AI coding agents as an attack surface in their own right. CSA's **AI Coding Agents as Attack Surface: MCP, Poisoning, and Miasma** (June 2026) documented MCP auto-execution flaws, repository and configuration poisoning, and the Miasma supply chain worm as converging threats against developer tooling [11]. The WebSec findings add a fourth vector to that analysis. Search-layer poisoning reaches the agent without the developer cloning, installing, or connecting anything hostile, and the defensive priorities from the earlier paper apply directly: limit auto-execution, isolate agent runtimes, and treat agent-consumed content as untrusted.

CSA's **Agentjacking: MCP Injection Hijacks AI Coding Agents** (June 2026) analyzed how instructions injected into Sentry error events, a data source the agent reads during routine debugging, could hijack MCP-connected coding agents into running attacker commands with developer privileges [12]. Search poisoning follows the same pattern of injecting through a trusted debugging workflow. The difference is that the hostile data arrives through a general-purpose web index and not a single integration. The mitigations in that note, especially human approval for command execution and credential isolation, carry over directly.

CSA's **Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks** (April 2026) examined how attackers register package names that code-generating models hallucinate [13]. The WebSec benchmark's "hallucination attacks" category, and its finding of install steps for packages that were never published, show that slopsquatting payloads can now arrive through search results as well as through model output [3]. Registry-side verification is therefore needed whichever way the package name entered the agent's context. CSA's **AI Coding Agents: An Unaudited Supply Chain Node** (July 2026) treats coding agents themselves as a supply chain dependency exposed to slopsquatting and prompt-injection-driven compromise [14]. This note's argument that the retrieval layer belongs within the software supply chain extends that framing to the search APIs agents depend on.

For control mapping, the **AI Controls Matrix (AICM) v1.1** offers relevant controls in its supply chain management, application and interface security, and threat and vulnerability management domains. These can be applied to search API selection, retrieval-content validation, and dependency verification [15]. Organizations threat-modeling agentic development pipelines can use **MAESTRO**, CSA's agentic AI threat modeling framework, to place the search tool as an external data dependency and analyze injection risks across agent layers [16].

References

[1] Levy, S., Lempert, S., and Hoffman, N. "Introducing WebSec: The first benchmark of how secure are web search providers for agents." AIR Research Lab, November 23, 2026.

[2] AIR Research Lab. "WebSec Benchmark." AIR, November 2026.

[3] Hoffman, N., Lempert, S., and Levy, S. "AI Coding Agents Are Getting Malicious Results From Web Search Indexes, WebSec Benchmark Finds." The Hacker News (contributed article), November 23, 2026.

[4] Mualem, E., and Granat, D. "SolanaCry." AIR Research Lab, October 29, 2026.

[5] Mualem, E., Hoffman, N., and Granat, D. "Web Hole." AIR Research Lab, October 23, 2026.

[6] Google. "AI threats in the wild: The current state of prompt injections on the web." Google Security Blog, April 23, 2026.

[7] Spracklen, J., et al. "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs." 34th USENIX Security Symposium, August 2025.

[8] Xu, X., et al. "The Long Con on the Web: Earn-then-Exploit Fraud Attacks against Coding Agents." arXiv:2611.00725, October 31, 2026.

[9] OWASP GenAI Security Project. "LLM01: Prompt Injection." OWASP Top 10 for LLM Applications, 2025.

[10] Open Source Security Foundation. "OSV: Open Source Vulnerabilities." OSV.dev, accessed November 24, 2026.

[11] Cloud Security Alliance. "AI Coding Agents as Attack Surface: MCP, Poisoning, and Miasma." CSA AI Safety Initiative, June 28, 2026.

[12] Cloud Security Alliance. "Agentjacking: MCP Injection Hijacks AI Coding Agents." CSA AI Safety Initiative, June 12, 2026.

[13] Cloud Security Alliance. "Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks." CSA AI Safety Initiative, April 19, 2026.

[14] Cloud Security Alliance. "AI Coding Agents: An Unaudited Supply Chain Node." CSA AI Safety Initiative, July 8, 2026.

© 2026 Cloud Security Alliance. AI-assisted Rapid Research.

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

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