Hidden Unicode Instruction Injection in AI Agent Skills: Invisible Adversarial Payloads in Tool Descriptions, Skill Files, and MCP Servers

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-10

Key Takeaways

The following findings summarize the primary security implications of hidden Unicode instruction injection for organizations deploying agentic AI systems.

- AI agents and coding assistants that consume tool descriptions, skill files, and Model Context Protocol (MCP) server metadata are vulnerable to a class of prompt injection attack in which adversarial instructions are encoded using invisible Unicode characters — specifically the Unicode Tags block (U+E0000–U+E007F) and zero-width characters — that are fully invisible in every standard user interface but are processed and obeyed by the underlying language model.
- Security researcher wunderwuzzi (Johann Rehberger) demonstrated on February 11, 2026, that production agent platforms — including Claude Code, GitHub Copilot, OpenAI Codex Skills, Google Gemini CLI, and the OpenClaw Hub — will execute arbitrary shell commands and curl requests embedded as invisible instructions inside agent skill markdown files, surviving any visual code review [1].
- The foundational Unicode Tags exploitation technique was disclosed by Riley Goodside on January 11, 2024, and independently demonstrated against Anthropic Claude by wunderwuzzi on February 8, 2024 [2][3]. Anthropic's initial response characterized the finding as having no security impact; nearly two years later, the technique is being exploited in production agentic skill and tool ecosystems [1].
- Tool Poisoning Attacks against MCP servers, first formally described by Invariant Labs researchers Luca Beurer-Kellner and Marc Fischer on April 1, 2025 [4], extend this threat to tool metadata: malicious MCP servers embed invisible instructions directly in tool descriptions that steal SSH keys, redirect email, and exfiltrate private chat histories — with attack success rates on prominent LLM agents reaching 72.8% in independent benchmarking [5].
- Academic research has quantified the broader adversarial Unicode attack surface: encoding-based attacks achieve 64–67% success against open-source LLMs, homoglyph substitution 42–58%, and steganographic jailbreaks using invisible characters reach a 92% average attack success rate while evading external safety detectors [6][7].

- No deployed defense fully neutralizes all variants. Security teams should implement Unicode character normalization and filtering at AI pipeline ingress points, apply CI/CD linting rules to reject bidirectional overrides and Tags-block codepoints, enforce source verification for all skill and MCP server installations, and disable auto-approval modes in agentic coding tools pending architectural improvements from vendors.

Background

The rapid adoption of agentic AI systems — platforms where large language models orchestrate tools, call external APIs, execute code, and consume structured skill or plugin definitions — has created an attack surface that traditional application security frameworks did not anticipate. Unlike a web application that processes structured inputs through a well-defined parser, an agentic AI system processes natural language, tool metadata, retrieved documents, and configuration files through a single inference pipeline that does not distinguish between trusted instructions and adversarial content embedded in data. This architectural characteristic is the root cause of indirect prompt injection, identified by OWASP as the primary variant of LLM01:2025, the highest-severity entry in the OWASP Top 10 for LLM Applications [8].

Hidden Unicode instruction injection represents a particularly severe subclass of indirect prompt injection because it defeats the primary human defense against prompt injection attacks: visual review. When an analyst inspects a tool description, a skill file, or a README, their eyes process rendered text — rendered with full awareness of Unicode's display rules, which specify that characters in the Tags block (U+E0000–U+E007F) and many zero-width characters carry no visual representation. The analyst sees only the legitimate, benign content. The language model, however, processes the raw token stream, where each invisible codepoint is a distinct, meaningful token. A well-crafted payload can therefore pass peer review, penetrate CI/CD pipelines that scan only for visible text patterns, survive packaging and distribution through skill marketplaces, and execute silently when the agent instantiates the skill.

The Unicode Tags block was created to serve a narrow technical purpose — language tagging within text streams — and its invisibility property is a documented Unicode standard behavior, not a bug in any particular implementation. This means the attack is not a vulnerability in Claude, ChatGPT, or any specific model in isolation. It is a structural property of how large language models tokenize and process text that was learned during training on corpora containing Unicode Tags codepoints, giving the model the ability to interpret them as meaningful content. Anthropic's initial closure of the vulnerability report in early 2024 reflected this framing — that the behavior is a model characteristic rather than a security defect in the traditional sense — but production exploitation in 2025 and 2026 has demonstrated that the practical security consequence is significant regardless of how the underlying behavior is classified.

Technical Analysis

The Unicode Tags Attack Primitive

The Unicode Tags block spans U+E0000 through U+E007F and mirrors the standard ASCII range: U+E0041 corresponds to capital A, U+E0061 to lowercase a, and so on through the full printable ASCII character set. Any arbitrary text string can be converted into its Tags-block equivalent, producing a sequence of codepoints that renders as nothing at all in web browsers, terminals, text editors, chat interfaces, and code review tools, but that an LLM tokenizer will split and process as a coherent, readable payload.

The transformation is trivial to implement. Security researcher wunderwuzzi released the ASCII Smuggler tool in January 2024, demonstrating encoding and decoding of arbitrary payloads [25][3]. A Python one-liner captures the core operation: each character in the cleartext is offset by 0xE0000. The resulting string occupies legitimate UTF-8 encoding space, passes as valid Unicode, and contains no codepoints that would trigger conventional injection-signature filters looking for phrases like "ignore previous instructions."

Cisco AI Security researcher Adam Swanda characterized the tokenizer behavior that enables the attack in a March 2024 technical analysis [9]: the LLM tokenizer splits the tagged text into its constituent codepoints and reconstructs the underlying payload, so the model "essentially re-builds the payload for you." The tagged characters and their untagged equivalents are processed through the same semantic pathways in the model, so the LLM's instruction-following response to a Tags-encoded directive is indistinguishable from its response to the same directive in plaintext.

The YARA and Python detection patterns needed to identify the attack are straightforward. The UTF-8 byte range \xF3\xA0\x80\x80 through \xF3\xA0\x81\xBF identifies all Tags-block codepoints. Any occurrence density above a trivial threshold (Swanda's published YARA rule flags more than ten matching sequences in a document) is a reliable indicator of encoded payload presence [9]. The challenge is not detection once the scanning tool is in place — it is that no standard toolchain currently applies this check to AI agent skill files, MCP tool description manifests, or model configuration repositories as a matter of course.

Zero-Width Character Injection

Distinct from the Tags-block technique, zero-width characters in the general Unicode character set present a complementary attack primitive. Characters including the Zero Width Space (U+200B), Zero Width Non-Joiner (U+200C), Zero Width Joiner (U+200D), and Zero Width No-Break Space (U+FEFF) are invisible in rendered text but present in the raw character stream. Knostic's October 2025 analysis documented the use of these characters in malicious AI coding assistant rules files — the `.cursorrules`, `.mdc`, and `.clinerules` configuration files that AI coding agents consume at session initialization [10]. By encoding instructions as binary patterns using U+200B for zero bits and U+200C for one bits (with U+200D as a terminator), an attacker can embed arbitrary text steganographically in what appears to be a blank line between two legitimate instructions.

The Promptfoo research team documented this encoding scheme in detail in April 2025, noting that LLMs process these characters as "distinct, valid Unicode characters in the input stream" [11]. An attacker who can insert or modify an AI coding assistant's rules file — through a supply chain compromise, a malicious open-source contribution, or a compromised community template repository — can deliver hidden instructions that will execute at the next session initialization, regardless of how carefully a developer reads the file's displayed content.

The GlassWorm incident reported by Knostic involved at least 35,800 affected IDE extension installations in which invisible characters were used to hide loader code, representing the first known supply chain attack of this type against AI coding assistant configuration infrastructure [10].

Bidirectional Unicode and Homoglyph Variants

The Unicode Bidirectional Override characters — specifically Right-to-Left Override (U+202E) and related control characters in the range U+202A–U+202E — have a longer history as security attack primitives in the context of filename spoofing and social engineering, but they also have application in AI prompt injection contexts. By reversing the visual display order of a string of text while leaving the underlying codepoint sequence unchanged, an attacker can create a payload that reads as one instruction to a human reviewer and a different instruction to a parser that operates on the raw codepoint sequence. OWASP's LLM Prompt Injection Prevention Cheat Sheet specifically identifies bidirectional override as an evasive technique requiring rejection or sanitization at AI pipeline ingress [12].

Homoglyph substitution exploits a different property of the Unicode standard: the existence of multiple characters in different scripts that are visually identical or nearly identical. Cyrillic contains eleven lowercase glyphs that are effectively indistinguishable from their Latin counterparts — Cyrillic а (U+0430) versus Latin a (U+0061) being the canonical example. An attacker who substitutes Cyrillic

characters for Latin equivalents in a prompt injection payload may evade keyword-based safety filters that match on the Latin byte sequences while the LLM tokenizer, trained on multilingual corpora, correctly reconstructs the intended semantic content. The Elastic Security Labs research published in September 2025 measured homoglyph attack success rates of 42.1–58.7% against open-source LLMs [6].

The practical defense for homoglyph attacks is Unicode normalization: NFKC normalization maps compatibility-equivalent characters to their canonical forms, collapsing Cyrillic lookalikes to their Latin equivalents before further processing. This is a well-understood technique in traditional application security contexts, but its application to AI pipeline input validation remains inconsistently implemented.

MCP Tool Poisoning: Invisible Instructions in Tool Metadata

The Model Context Protocol, introduced by Anthropic in November 2024 as a standardized interface for connecting LLM agents to external tools, services, and data sources, has created a new delivery vector for Unicode-based hidden instruction injection. MCP servers expose tool definitions consisting of a name, a description, and a schema for input parameters. These descriptions are transmitted to the connected LLM agent and inform its reasoning about when and how to use each tool. Critically, tool descriptions are displayed to users only in summarized or abbreviated form in most MCP client implementations — the full text of the description is consumed by the model, not by the human operator.

Invariant Labs formally characterized Tool Poisoning Attacks (TPAs) in April 2025, demonstrating that malicious MCP servers can embed hidden directives in tool descriptions that instruct the LLM to perform actions entirely unrelated to the tool's stated function [4]. Their proof-of-concept against Cursor Agent used a tool nominally performing simple arithmetic that contained embedded instructions directing the agent to read `~/.cursor/mcp.json` and transmit SSH private keys to an attacker-controlled endpoint. A follow-up demonstration showed a poisoned tool description instructing the agent to redirect all outgoing email to an attacker-controlled address — with the agent executing the redirection faithfully while displaying only the legitimate email operation to the user.

The MCPTox benchmark study, published in August 2025, evaluated 20 prominent LLM agents against 45 real-world MCP servers and 353 authentic tools, finding attack success rates as high as 72.8% with the o1-mini model [5]. The study reached a counterintuitive conclusion: more capable models tend to be more vulnerable, not less, because the attack exploits superior instruction-following fidelity. Models that are better at following instructions follow the attacker's instructions more reliably. The highest observed refusal rate across all tested models — achieved by Claude-3.7-Sonnet — was less than 3%.

The February 2026 wunderwuzzi research on agent skill files extended the MCP tool poisoning threat model specifically to include hidden Unicode Tags payloads [1]. Where earlier Tool Poisoning Attack demonstrations embedded plaintext hidden instructions that could theoretically be surfaced by rendering the full tool description, the Unicode Tags variant is invisible even under full visual inspection of the raw metadata. The research demonstrated execution of arbitrary shell commands and curl calls embedded as invisible instructions in a legitimate OpenAI "security-best-practices" skill file, with successful exploitation confirmed against Claude Code (with auto-approval enabled), GitHub Copilot with Claude 4.5, OpenAI Codex Skills, Google Gemini CLI, and the OpenClaw Hub.

Threat Scenarios in Production Environments

Supply Chain Injection via Skill Marketplaces and Template Repositories

Community skill repositories, skill marketplaces, and open-source template repositories represent the highest-leverage attack surface for hidden Unicode instruction injection. An adversary who contributes a malicious skill to an open-source repository — or who compromises a legitimate skill file in transit — can ensure that every developer who installs that skill executes hidden instructions at their next agent session. The Knostic analysis of GlassWorm established that this threat is not hypothetical: 35,800 confirmed installations of a compromised IDE extension carrying invisible character payloads represent a supply chain attack of a scale that would merit a critical advisory in any other software ecosystem [10].

The characteristics of skill and configuration files make them particularly susceptible to this attack. They are markdown and text files, not executable binaries, and they are therefore afforded a lower level of scrutiny in typical security review processes. Supply chain controls that scan for malicious code in compiled dependencies do not apply. Code review tools surface the rendered content, not the raw codepoint stream. The attack is designed precisely to defeat the review mechanisms that developers naturally apply.

MCP Server Compromise and Rug-Pull Attacks

Elastic Security Labs documented a second critical variant: the rug-pull attack, in which an MCP server initially presents benign tool descriptions to gain user and organizational approval, then updates those descriptions to include malicious hidden instructions after approval has been granted [13]. Existing MCP client implementations do not implement cryptographic pinning or hash verification of tool descriptions,

so a malicious update to an approved MCP server is transparent to the connecting agent. The Elastic research also documented tool name collision, in which a malicious server's tool description overrides instructions from trusted tools registered by legitimate servers in the same MCP session.

The August 2024 Slack AI data exfiltration incident, in which indirect prompt injection delivered through private channel content caused the Slack AI summarization agent to transmit sensitive conversations to attacker-controlled addresses, established the real-world consequence profile of this attack class [14]. The December 2025 web-based indirect prompt injection incident documented by Palo Alto Networks Unit 42 — involving a real-world ad-review bypass using multiple concealment techniques including invisible Unicode — confirmed that attackers are actively combining obfuscation methods and targeting AI-mediated workflows with high-severity fraud objectives [15].

Agentic Coding Assistants and Remote Code Execution

CVE-2025-53773, disclosed by wunderwuzzi on August 12, 2025, demonstrated the highest-severity consequence class: remote code execution via prompt injection in GitHub Copilot [16]. The exploit chain used injected instructions to modify `.vscode/settings.json`, enabling auto-approval mode that eliminated confirmation requirements for shell command execution, followed by OS-targeted terminal commands. The same research confirmed that Unicode-based invisible instructions could trigger portions of this exploit chain. Microsoft patched the vulnerability in August 2025.

CVE-2025-32711 (EchoLeak), disclosed in June 2025 and analyzed by Pavan Reddy and Aditya Gujral, demonstrated zero-click data exfiltration from Microsoft 365 Copilot through a crafted email containing hidden instructions that evaded Microsoft's Cross Prompt Injection Attempt classifier [17]. The attack required no user interaction beyond normal Copilot usage and represented, in the authors' characterization, "the first known case of a prompt injection being weaponized to cause concrete data exfiltration in a production AI system."

Academic Research Landscape

The academic community has characterized the Unicode adversarial attack surface with increasing rigor since the initial 2024 disclosures. Three research threads are particularly relevant to practitioners.

Geng et al.'s May 2025 paper introduced StegoAttack, a steganography-based jailbreak that hides harmful queries within benign, semantically coherent text and instructs the model to extract and respond using encrypted output formats [7]. StegoAttack achieved a 92% average attack success rate across

four safety-aligned commercial LLMs while maintaining less than 1% degradation in performance against the Llama Guard external safety detector — demonstrating that invisible encoding defeats not only human visual review but also dedicated external safety classifiers.

Sarabamoun's August 2025 systematic study from Capital One evaluated character-level adversarial attacks across seven open-source LLMs using 2,800+ attack instances [6]. Encoding-based attacks (Base64, hexadecimal, ROT-n) achieved 64.3–67.1% success rates; Unicode control characters including U+200B, U+200C, and U+202E achieved 52–54%; and homoglyph substitution achieved 42.1–58.7%. The research identified a "size-robustness decoupling" phenomenon: the 7B Mistral model was most vulnerable at 85.5%, while a 20B model achieved only 18.5% vulnerability, demonstrating that parameter count is not a reliable predictor of resistance to character-level adversarial attacks.

The December 2025 arxiv study analyzing Unicode text watermarking methods tested ten format-based watermarking approaches against multiple LLMs including GPT-5, GPT-4o, Llama 3.3, Claude Sonnet 4, and Gemini 2.5, confirming that whitespace, control characters, and zero-width characters occupy a well-understood niche in LLM token processing that watermarking and adversarial injection techniques exploit through the same underlying mechanism [18].

Trend Micro's January 2025 empirical testing using NVIDIA's Garak adversarial evaluation framework against AWS Bedrock models found attack success rates of 87.5% against Claude 3.5 Sonnet and 12.5% against Claude 3 Opus [19]. The wide variation between Claude model generations highlights that Unicode Tag susceptibility is not uniform even within a single vendor's product line, and that defensive measures implemented in the training process can meaningfully reduce — though not eliminate — vulnerability.

Framework Alignment

MAESTRO Threat Model

The Cloud Security Alliance's MAESTRO framework (Multi-Agent Environment, Security, Threat, Risk, and Outcome), published in February 2025 by Ken Huang of DistributedApps.ai, provides the most directly applicable threat modeling structure for the attack class described in this note [20]. MAESTRO's seven-layer reference architecture isolates the Tool/Skill Layer as a distinct threat surface exposed to compromised external integrations — a categorization that maps precisely to MCP server poisoning and malicious skill file injection. The framework's T3.2 (Prompt Injection) mitigation recommendations include input sanitization with removal of special characters that could be interpreted as instructions, clear separation of user input from system instructions, instruction hardening, and output validation.

These mitigations are necessary but insufficient for the Unicode Tags variant: removing "special characters" without specifically targeting the Tags-block codepoint range will not remove a well-crafted invisible payload.

OWASP LLM Top 10:2025

OWASP classifies prompt injection as LLM01:2025, the highest-severity entry in the LLM Application security list, updated April 2025 [8]. The 2025 revision explicitly acknowledges invisible prompt injection techniques including hidden Unicode payloads. The companion OWASP Prompt Injection Prevention Cheat Sheet [12] specifically calls for rejection or flagging of bidirectional Unicode overrides and non-printing Unicode characters. Homoglyph substitution and encoding obfuscation are listed as evasion techniques requiring dedicated defense at the input validation layer.

MITRE ATLAS

MITRE ATLAS (Adversarial Threat Landscape for Artificial-Intelligence Systems) catalogs prompt injection as technique AML.T0051, classified under the Initial Access tactic [21]. The framework, which as of October 2025 comprises 15 tactics, 66 techniques, and 46 sub-techniques (note: counts reflect the October 2025 snapshot and should be verified against the current knowledge base, as the framework is updated continuously), maps AML.T0051 to both direct and indirect injection sub-techniques and cross-references MITRE ATT&CK T1027 (Obfuscated Files or Information) as the relevant obfuscation technique — the precise mapping that applies to Unicode Tags encoding. ATLAS serves as the authoritative threat taxonomy for AI-targeted attacks and is the recommended framework for classifying hidden Unicode injection in organizational threat modeling exercises.

NIST AI RMF and Generative AI Profile

The NIST AI Risk Management Framework's Generative AI Profile, publicly announced in July 2024 and finalized as NIST AI 600-1 [22], identifies prompt injection as a critical GAI vulnerability class and recommends governance through the RMF's Govern, Map, Measure, and Manage functions. NIST's position is that prompt injection cannot be fully solved within current architectures and requires defense-in-depth, continuous red-teaming, runtime monitoring, strict privilege minimization, and human-in-the-loop controls for high-stakes actions — a posture consistent with the findings of the academic and practitioner research surveyed in this note.

Defense Recommendations

The security controls for hidden Unicode instruction injection span multiple layers of the AI pipeline and must address the four primary delivery vectors — agent skill files, MCP tool descriptions, retrieved documents and web content, and user-supplied inputs — with independent controls at each layer.

At the input sanitization layer, organizations should apply Unicode normalization (NFKC form) to all text ingested by AI pipelines before tokenization, stripping or rejecting codepoints in the Tags block (U+E0000–U+E007F), zero-width characters (U+200B–U+200D, U+FEFF), and bidirectional override controls (U+202A–U+202E). The detection regex `[\x00-\x1F\x7F-\x9F\u200B-\u200D\uFEFF\u202A-\u202E]` published by Knostic provides a practical grep-based starting point for scanning repositories and configuration files [10]. The YARA rule published by Cisco's Adam Swanda (matching UTF-8 pattern \xF3\xA0[0x00-0x02]??) provides analogous coverage for endpoint and pipeline security tooling [9].

At the build and supply chain layer, CI/CD pipelines should incorporate Unicode linting that rejects any skill file, MCP server manifest, or AI configuration file containing Tags-block codepoints or anomalous zero-width character densities. This check should be applied to all vendored dependencies, community-sourced skill packages, and contributions to shared configuration repositories. The same analysis should be applied to third-party MCP servers before they are approved for organizational use. Version pinning with cryptographic hash verification of tool descriptions, as recommended by Invariant Labs [4], prevents rug-pull attacks in which a trusted MCP server is silently updated to include malicious instructions.

At the agent configuration layer, auto-approval modes in agentic coding tools — the `chat.tools.autoApprove` setting in VS Code and equivalent settings in other environments — should be disabled by default and subject to explicit security review before enablement. As the CVE-2025-53773 exploit demonstrated, auto-approval eliminates the human confirmation step that would otherwise catch an agent executing unexpected shell commands [16]. Organizations should also adopt the minimum-privilege principle for agent tool access, limiting the actions available to an agent to those strictly necessary for its designated function.

At the detection and response layer, Lasso Security's claude-hooks open-source project, released January 2026, provides a PostToolUse hook mechanism for Claude Code that intercepts tool outputs before the model processes them and applies 50+ detection patterns across injection categories including encoding obfuscation and homoglyph substitution [23]. The `aid` scanner referenced in

wunderwuzzi's February 2026 research specifically targets Unicode Tag sequences in skill files and agent configuration content [1]. AWS has published guidance on implementing Unicode character smuggling defenses in LLM applications using Amazon Bedrock Guardrails [24].

CSA Resource Alignment

This research note addresses threats that map to multiple active Cloud Security Alliance frameworks and working group outputs. The MAESTRO framework provides the primary threat modeling structure, with the Tool/Skill Layer (Layer 3) threats directly applicable to MCP tool poisoning and skill file injection. The CSA AI Safety Initiative's Top Threats to Cloud Computing and AI 2026 report identifies supply chain compromise and indirect prompt injection as tier-one threats consistent with the attack class documented here. Organizations should implement CSA's AI Controls Matrix (AICM) — a superset of the Cloud Controls Matrix (CCM) designed specifically for cloud-based AI systems — as their primary governance framework for agentic AI deployments. AICM's controls for AI application security, model input validation, and AI supply chain integrity directly address the unicode injection and tool poisoning vectors documented in this note. The Application and Interface Security (AIS) control domain requirements should be applied to AI pipeline input validation, treating skill files and MCP server tool descriptions as application inputs subject to the same validation requirements as web application inputs.

CSA's CISA-aligned guidance on Zero Trust Architecture is directly relevant: the principle of never-implicit-trust applies to AI agent tool and skill configurations with the same force it applies to network access decisions. No skill or MCP server should be treated as trusted by virtue of its source alone — cryptographic provenance, behavioral sandboxing, and runtime monitoring are required components of a defensible agentic AI deployment. Organizations using AICM alongside the Security, Trust, Assurance, and Risk (STAR) program can evaluate AI tool and agent providers against these controls as part of their vendor assessment process.

Limitations and Uncertainties

Several aspects of this threat landscape involve genuine uncertainty that practitioners should account for in their risk assessments.

Vendor-specific mitigation status is incompletely documented. Anthropic's initial 2024 response to the Tags-block vulnerability report characterized it as not a security issue; subsequent behavior as of early 2026 is not definitively established. Trend Micro's empirical data showed an 87.5% attack success rate against Claude 3.5 Sonnet, but model behavior changes with version updates, and current production model susceptibility may differ. Organizations should conduct their own empirical testing using tools such as NVIDIA Garak before drawing conclusions about their specific deployment's current exposure.

The MCPTox benchmark finding that more capable models are more vulnerable is a significant and counterintuitive result that has not been fully replicated across independent research teams. It is plausible that the observed correlation between capability and vulnerability reflects the specific attack scenarios tested rather than a general principle.

The claim that Claude-3.7-Sonnet has a maximum refusal rate under 3% against Tool Poisoning Attacks (from MCPTox) should be interpreted carefully: the study tested specific attack configurations, and model behavior on out-of-distribution attacks not represented in the benchmark may differ.

References

[1] wunderwuzzi (Johann Rehberger). "Scary Agent Skills: Hidden Unicode Instructions in Skills...And How To Catch Them." Embrace The Red. February 11, 2026. https://embracethered.com/blog/posts/2026/scary-agent-skills/

[2] Riley Goodside. Unicode Tags prompt injection proof-of-concept. Twitter/X. January 11, 2024. (Original post; referenced in multiple secondary sources including [3] and [9].)

[3] wunderwuzzi (Johann Rehberger). "Hidden Prompt Injections with Anthropic Claude." Embrace The Red. February 8, 2024. https://embracethered.com/blog/posts/2024/claude-hidden-prompt-injection-ascii-smuggling/

[4] Luca Beurer-Kellner and Marc Fischer. "MCP Security Notification: Tool Poisoning Attacks." Invariant Labs. April 1, 2025. https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks

[5] MCPTox benchmark study. "MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers." arXiv:2508.14925. August 2025. https://arxiv.org/html/2508.14925v1

[6] Ephraiem Sarabamoun (Capital One). "Special-Character Adversarial Attacks on Open-Source Language Models." arXiv:2508.14070. August 12, 2025. https://arxiv.org/html/2508.14070v1

[7] Jianing Geng, Biao Yi, Zekun Fei, Tongxi Wu, Lihai Nie, Zheli Liu. "When Safety Detectors Aren't Enough: A Stealthy and Effective Jailbreak Attack on LLMs via Steganographic Techniques." arXiv:2505.16765. May 22, 2025. https://arxiv.org/abs/2505.16765

[8] OWASP Gen AI Security Project. "LLM01:2025 Prompt Injection." OWASP Top 10 for LLM Applications 2025. Originally published April 10, 2024; last modified April 17, 2025. https://genai.owasp.org/llmrisk/llm01-prompt-injection/

[9] Adam Swanda (Robust Intelligence / Cisco). "Understanding and Mitigating Unicode Tag Prompt Injection." Cisco AI Blog. March 12, 2024. https://blogs.cisco.com/ai/understanding-and-mitigating-unicode-tag-prompt-injection

[10] Knostic Team. "Zero Width Unicode Characters: the Risks you Can't See." Knostic Blog. October 21, 2025. https://www.knostic.ai/blog/zero-width-unicode-characters-risks

[11] Asmi Gulati. "The Invisible Threat: How Zero-Width Unicode Characters Can Silently Backdoor Your AI-Generated Code." Promptfoo Blog. April 10, 2025. https://www.promptfoo.dev/blog/invisible-unicode-threats/

[12] OWASP. "LLM Prompt Injection Prevention Cheat Sheet." OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/LLM_Prompt_Injection_Prevention_Cheat_Sheet.html

[13] Carolina Beretta, Gus Carlock, and Andrew Pease. "MCP Tools: Attack Vectors and Defense Recommendations for Autonomous Agents." Elastic Security Labs. September 19, 2025. https://www.elastic.co/security-labs/mcp-tools-attack-defense-recommendations

[14] Slack AI data exfiltration incident. August 2024. Referenced in multiple secondary sources including Lakera indirect prompt injection analysis (https://www.lakera.ai/blog/indirect-prompt-injection) and Unit 42 AI agent prompt injection research.

[15] Beliz Kaleli, Shehroze Farooqi, Oleksii Starov, and Nabeel Mohamed. "Fooling AI Agents: Web-Based Indirect Prompt Injection Observed in the Wild." Palo Alto Networks Unit 42. March 3, 2026. https://unit42.paloaltonetworks.com/ai-agent-prompt-injection/

[16] wunderwuzzi (Johann Rehberger). "GitHub Copilot: Remote Code Execution via Prompt Injection (CVE-2025-53773)." Embrace The Red. August 12, 2025. https://embracethered.com/blog/posts/2025/github-copilot-remote-code-execution-via-prompt-injection/

[17] Pavan Reddy and Aditya Sanjay Gujral. "EchoLeak: The First Real-World Zero-Click Prompt Injection Exploit in a Production LLM System." arXiv:2509.10540. September 6, 2025. CVE-2025-32711. https://arxiv.org/html/2509.10540v1

[18] "Security and Detectability Analysis of Unicode Text Watermarking Methods Against Large Language Models." arXiv:2512.13325. December 2025. https://arxiv.org/html/2512.13325

[19] Ian Ch Liu. "Invisible Prompt Injection: A Threat to AI Security." Trend Micro Research. January 22, 2025. https://www.trendmicro.com/en_us/research/25/a/invisible-prompt-injection-secure-ai.html

[20] Ken Huang (DistributedApps.ai). "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance. February 6, 2025. https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro

[21] MITRE ATLAS. Technique AML.T0051: Prompt Injection. https://atlas.mitre.org/

[22] NIST. "Artificial Intelligence Risk Management Framework: Generative AI Profile." NIST AI 600-1. July 2024. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[23] Or Oxenberg and Eliran Suisa (Lasso Security). "Detecting Indirect Prompt Injection in Claude Code with Lasso." Lasso Security Blog. January 7, 2026. https://www.lasso.security/blog/the-hidden-backdoor-in-claude-coding-assistant

[24] AWS Security Blog. "Defending LLM Applications against Unicode Character Smuggling." Amazon Web Services. https://aws.amazon.com/blogs/security/defending-llm-applications-against-unicode-character-smuggling/

[25] wunderwuzzi (Johann Rehberger). "ASCII Smuggler Tool: Crafting Invisible Text and Decoding Hidden Codes." Embrace The Red. January 14, 2024. https://embracethered.com/blog/posts/2024/hiding-and-finding-text-with-unicode-tags/