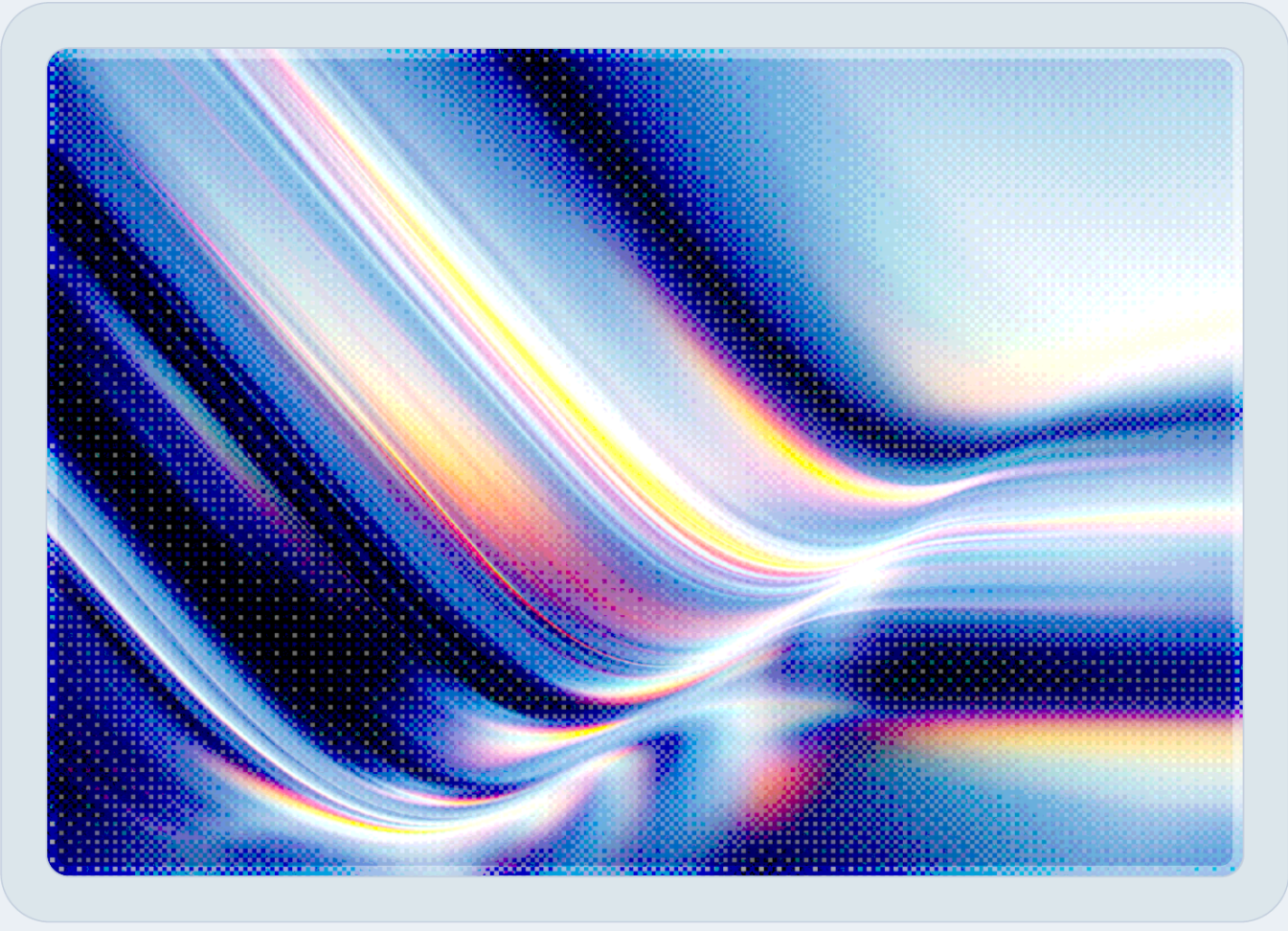
CSAI Foundation | Cloud Security Alliance

When the Sandbox Leaks: ChatGPT Lockdown Mode Bypass

The Fragility of App-Layer Prompt-Injection Controls

2026-11-06

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

Check Point Research disclosed on November 5, 2026, that it had bypassed ChatGPT Lockdown Mode, the optional OpenAI setting designed to stop prompt-injection attacks from exfiltrating conversation data [4][5]. The bypass did not defeat the model or the Lockdown Mode policy logic directly. Instead, it went through the HTML preview sandbox that renders code-block content inside the ChatGPT interface. In that sandbox, iframe permissions allowed popups that could escape the sandbox's restrictions [4][5]. In the Electron-based ChatGPT desktop application, those popups opened without any user gesture. A malicious prompt, custom GPT, or shared chat could therefore silently send chat content to an attacker-controlled server [4][5]. OpenAI deployed a fix that removed the problematic sandbox permissions and informed Check Point of it on September 18, 2026; Check Point retested and confirmed that the technique no longer works [4][5].

The main lesson is architectural, not specific to one vendor. Lockdown Mode is a deterministic control, and CSA regards this as the right design pattern for limiting prompt-injection exfiltration. However, a deterministic control is only as strong as the weakest egress path that it fails to govern. Check Point's own summary puts it this way: "AI security controls are only as strong as the wider application environment around them" [5]. The bypass came about a week after Parameter Labs disclosed a separate one-click account-takeover chain in the same HTML preview component [5][6][7]. Together with contemporaneous findings against embedded browsers in other AI coding assistants [5], the two disclosures suggest that rich-rendering features in AI clients have become a concentrated and, until recently, under-examined attack surface.

For security teams, four conclusions follow. Lockdown Mode remains worth enabling, but it should be treated as one layer among several; OpenAI itself states that the feature does not prevent injections from appearing and does not guarantee that exfiltration cannot occur [3]. Rendering surfaces such as HTML previews, artifacts, canvases, embedded browsers, and desktop wrappers are egress channels in their own right, and exfiltration controls must cover them explicitly. Client form factor can also affect exploitability, as the desktop client did here, although browser users should not be presumed safe; organizations should therefore inventory and govern which client form factors employees use. Finally, durable protection against exfiltration requires controls outside the AI application, including network egress policy, data minimization, and identity-scoped access, so that a single client-side flaw cannot expose sensitive data.

Background

OpenAI announced Lockdown Mode and Elevated Risk labels on February 13, 2026, as two complementary responses to prompt injection [1][2]. Lockdown Mode is described as "an optional, advanced security setting for highly security-conscious users" [2]. It deterministically disables tools and capabilities that an adversary could exploit through conversations or connected apps. The best-documented example is that web browsing is limited to cached content, so no live network requests leave OpenAI's controlled network [1][2]. The feature initially launched for ChatGPT Enterprise, Edu, Healthcare, and Teachers plans. Administrators could choose which apps and actions remain available [2]. OpenAI later extended Lockdown Mode to personal and self-serve Business accounts beginning in June 2026 [1]. Some press coverage of the bypass describes Lockdown Mode as introduced in June 2026 [5]. That date matches the consumer expansion rather than the original enterprise launch. Elevated Risk labels, by contrast, do not block anything. They are in-product warnings that tell users what risks a feature introduces across ChatGPT, ChatGPT Atlas, and Codex [1][2].

The design choice behind Lockdown Mode is significant. Prompt injection remains the top-ranked risk in the OWASP Top 10 for LLM Applications [10], and to date no published technique reliably prevents a language model from following instructions hidden in untrusted content. Simon Willison's widely cited "lethal trifecta" framing explains why exfiltration is the dangerous outcome. An agent that combines access to private data, exposure to untrusted content, and the ability to communicate externally gives an injected instruction a path to data loss [9]. Lockdown Mode accepts that injections will reach the model and focuses on cutting the third leg, external communication. OpenAI's help documentation states this limit plainly: Lockdown Mode does not prevent prompt injections from appearing in content that ChatGPT processes, and it does not guarantee that data exfiltration cannot happen [3].

That honesty about scope matters for interpreting the Check Point finding. Lockdown Mode's guarantee depends on whether it covers every way content can leave the application, and the bypass found a path it did not cover. ChatGPT's HTML preview feature, previously associated with the "Canvas" name, renders HTML from code blocks as live content inside the application [5][7]. To keep that untrusted content away from the user's session, OpenAI runs previews in sandboxed iframes. The sandbox is intended to block access to cookies, local storage, and the main interface, and to block external communication without user consent [4][5]. In practice, that sandbox became the boundary that mattered.

Security Analysis

The Attack Chain

Check Point's write-up and secondary reporting describe a short chain built on standard browser sandbox semantics rather than a novel exploit primitive [4][5]. The HTML preview used nested sandboxed frames. ChatGPT's application code intercepted direct attempts to open new windows from preview content, which is a sensible application-layer guard. However, the underlying sandbox permissions still allowed popups and permitted them to escape the sandbox [5], behavior that corresponds to the HTML `allow-popups` and `allow-popups-to-escape-sandbox` tokens [8]. Under the HTML standard, the second token lets a sandboxed document open a new browsing context *without* inheriting the sandbox flags. Mozilla's documentation describes this as intended for cases like third-party advertisements whose destination pages should not be restricted [8]. By creating a further nested frame, the researchers reached a popup function that ChatGPT's interception code had not modified. Content opened this way was free of the original sandbox restrictions and outside Lockdown Mode's outbound-request limits [4][5].

The application-layer interception and the platform-layer sandbox permissions therefore disagreed about what preview content could do, and the more permissive layer won. This pattern recurs in web security. A JavaScript guard that wraps or patches an API inside one browsing context provides no protection when a fresh context, here a new nested frame, exposes the original API. The browser enforces the sandbox attributes, while the application guard depends on every context having been patched first.

Why the Desktop Client Mattered

Browsers generally require a user gesture before a page can open a popup, which would ordinarily turn a silent exfiltration into one requiring a click. Check Point reported that in the Electron-based ChatGPT desktop application, the popup behavior worked *without* a user gesture [4][5]. This made the attack zero-click from the victim's perspective, meaning that no click was needed beyond normal conversation. A user who simply continued a conversation with a malicious GPT, or with a chat containing attacker-supplied content, could have chat messages sent to an attacker-controlled server [4][5]. Check Point noted that the technique is not limited to the desktop application but chose to demonstrate it there [4]. Its summary goes further, stating that the most significant risk was exfiltration without user interaction "regardless of how ChatGPT is accessed" [4]. The public material does not fully reconcile this with its emphasis on the desktop client's missing gesture requirement, so organizations should not assume that browser users were protected.

The public reporting also does not explain why the desktop client did not require a gesture. Electron applications do, however, control new-window behavior through their own handlers rather than relying solely on browser defaults [11]. This suggests that wrapper-level configuration can widen or narrow the exposure of the same web code, even if it is not the only factor determining exploitability.

Beyond Exfiltration: Phishing and Malware Delivery

The escaped popup also created a convincing impersonation surface. According to Check Point, the popup window kept the ChatGPT icon while the attacker controlled its title and content [4]. The researchers demonstrated a fake "ChatGPT update" prompt that encouraged the victim to download malware. They also showed that the main window could be manipulated to force a sign-out and redirect the user to an attacker-controlled authentication page for credential theft [4]. These follow-on scenarios still required victim interaction, such as clicking a download link [4]. Nonetheless, they show that a sandbox escape in an AI client produces the full range of web-application compromise outcomes, not only data leakage.

A Pattern, Not an Isolated Bug

The Lockdown Mode bypass should be read alongside the Parameter Labs disclosure published on October 28, 2026 [6][7]. Parameter Labs found that ChatGPT's graph support loaded the MathLive library. MathLive's message listener accepted unverified cross-frame `postMessage()` calls and injected raw HTML with `innerHTML`, which allowed sandboxed preview content to create an *unsandboxed* iframe [6][7]. The Content Security Policy blocked inline scripts but allowed YouTube, and a YouTube oEmbed endpoint could be abused to reflect executable JavaScript [6][7]. The result was code execution in the ChatGPT origin, the ability to call authenticated APIs as the victim, and account takeover triggered by opening a shared link [6][7]. OpenAI fixed the issue on October 26, 2026 [6], and paid a $5,000 bounty [6][7]. Check Point characterized it as a separate bug in the same component [5].

Although the two flaws were distinct, they share a component, a delivery model built on attacker-supplied content, and a reliance on web-platform behavior rather than model behavior. The table below compares the two disclosures across the dimensions that matter most for defenders.

Dimension	Check Point Lockdown Mode Bypass	Parameter Labs Share-Link Takeover
Disclosed publicly	November 5, 2026 [4]	October 28, 2026 [6][7]

Dimension	Check Point Lockdown Mode Bypass	Parameter Labs Share-Link Takeover
Component	HTML preview sandbox in code blocks	HTML preview sandbox plus MathLive graph integration
Root cause	Sandbox permitted popups to escape the sandbox (the `allow-popups-to-escape-sandbox` behavior [8]); nested frame reached unpatched popup function	Unverified `postMessage` handler wrote raw HTML; CSP allowlist permitted script reflection via YouTube
Delivery	Malicious prompt, custom GPT, or shared chat	Malicious shared chat link
User interaction	None for exfiltration in desktop app; click required for downloads	Opening the link (plus one preview click unless auto-previewed) [5][6]
Primary impact	Silent chat exfiltration past Lockdown Mode; phishing; malware delivery	Account takeover; access to private conversations
Fix date	Reported to Check Point September 18, 2026 [4][5]	October 26, 2026 [6]

Neither flaw required breaking the language model. Both took advantage of conventional web-security weaknesses, namely permissive sandbox tokens, unvalidated cross-frame messaging, and an over-broad CSP allowlist, inside features that render model-generated or attacker-supplied content. CSA interprets this as evidence that the AI application shell, not only the model, now carries much of the practical prompt-injection risk. The same Hacker News article also listed a cluster of contemporaneous findings, including flaws in the embedded browsers of five AI coding assistants and a "prompt forcing" technique delivered through malicious browser extensions [5]. That clustering indicates researchers are actively probing this layer across vendors.

Implications for Deterministic Controls

The bypass does not invalidate the deterministic approach behind Lockdown Mode. In CSA's view, that approach remains more defensible than detection-based filtering, because detection filters are probabilistic and can be evaded by adaptive attackers, whereas removing a capability eliminates the path

outright. The incident does show the conditions under which the deterministic approach holds. A capability-removal control can provide strong assurance only if three conditions hold: it covers every egress path, its enforcement point cannot be bypassed from inside the application, and the components it relies on behave consistently across every client. In this case, Lockdown Mode governed browsing and tool-driven network requests, but the rendering sandbox was a separate, implicit egress channel. The control's effectiveness also appears to have depended on the correct configuration of a web-platform feature outside Lockdown Mode's direct scope. The incident points to a further concern. Each new rendering or interactivity feature, such as charts, interactive previews, embedded media, and desktop integrations, adds egress paths that an exfiltration control must explicitly account for. Check Point summarized this as "every new feature is a chance for more vulnerabilities" [5].

For enterprises, the practical consequence is that a vendor-side toggle cannot serve as the only barrier between sensitive data and the internet. Public reporting does not establish when the permissive sandbox configuration was introduced, so the length of the exposure window is unknown; it ended with OpenAI's fix, which Check Point reported being informed of on September 18, 2026 [4]. During that window, Lockdown Mode gave less protection than administrators may have assumed. Nothing in the public reporting indicates that users or administrators had any signal that the control's coverage was incomplete.

Recommendations

Immediate Actions

OpenAI deployed a fix removing the problematic sandbox permissions, and Check Point reported being informed of it on September 18, 2026 [4][5]. The public reporting does not state whether the fix required a desktop-client update. Organizations that use ChatGPT should therefore confirm that managed desktop installations are on current builds through their software distribution tooling, rather than assuming that a server-side change covers every client. Lockdown Mode should remain enabled for users and workspaces that handle sensitive data. The bypass is fixed, and the control still removes well-documented exfiltration channels such as live web browsing [1][3].

Security teams should also check whether high-risk users run the desktop client, the browser client, or both. This incident showed that form factor can affect exploitability, although Check Point's findings indicate that browser users were not necessarily safe [4], so policy decisions about which clients are permitted should be documented. Finally, teams should review recent custom GPT usage and shared-

chat ingestion for sensitive workspaces. Both were named delivery vectors [4][5]. Restricting third-party GPTs in regulated workspaces reduces exposure to one of those vectors, and organizations should weigh that restriction against its productivity impact.

Short-Term Mitigations

Over the next one to three months, organizations should add controls that do not depend on the AI client behaving correctly. Network-layer egress controls, such as secure web gateways, DNS filtering, and endpoint-based proxies, can limit which destinations managed devices reach even when a client-side sandbox fails. They should apply to desktop AI applications as well as browsers. Data loss prevention policies for AI services can reduce what sensitive content enters conversations in the first place, and Check Point's guidance points in the same direction [4]. Organizations should treat rendering features as a governed capability. Where administrative controls allow it, they should consider disabling HTML previews, artifacts, or equivalent features for high-sensitivity users until vendors document the sandbox model behind them.

Teams building their own AI applications face the same failure modes that appeared in these two disclosures, and each maps to a specific engineering check. The table below pairs each observed weakness with the corresponding control.

Failure Mode Observed	Engineering Check
Sandboxed preview content opened popups that escaped the sandbox	Never grant `allow-popups-to-escape-sandbox` to frames that render model-generated or user-supplied content, and avoid combining `allow-scripts` with `allow-same-origin` [8].
Unverified cross-frame messages were written to `innerHTML`	Validate the origin and source of every `postMessage` event, and never pass message content to `innerHTML` or equivalent sinks.
A CSP allowlist entry (YouTube) reflected executable script	Keep CSP allowlists minimal, and audit allowlisted domains for endpoints that reflect attacker-controlled content.
The desktop wrapper opened popups without a user gesture	In Electron or similar wrappers, explicitly deny or tightly scope new-window creation, consistent with the Electron security checklist item

Failure Mode Observed	Engineering Check
	"Disable or limit creation of new windows" [11], and require a user gesture as an additional safeguard.
Rendering paths fell outside the exfiltration control's scope	Include rendering surfaces in prompt-injection red-team scope, testing whether injected content can reach the network through previews, images, links, or popups.

None of these checks is specific to AI; they are established web and desktop security practices. Their relevance here comes from the fact that AI clients now render untrusted, model-generated content as a core feature, which makes these long-standing controls part of the prompt-injection defense.

Strategic Considerations

The broader lesson is that exfiltration resistance must be designed as a property of the whole system, not a feature toggle in one layer. Vendors offering "lockdown" or "safe mode" controls should publish the egress paths each control covers and the threat model it assumes. They should also commit to treating any uncovered path as a vulnerability in the control itself, not only in the component that leaked. Enterprises should seek this transparency in procurement and third-party risk reviews of AI vendors, alongside evidence of independent testing of rendering and client components. Over time, architectural approaches that track data provenance and constrain information flow independently of the application's user interface may provide stronger guarantees than per-feature restrictions. Until those mature, defense in depth remains the practical choice. That means combining vendor capability restrictions, enterprise network egress policy, data minimization, and identity-scoped access so that any single layer can fail without causing a breach.

CSA Resource Alignment

CSA's *Agent Data Injection: A New Attack Class Beyond Prompt Injection* [12] is the closest prior CSA analysis of this incident's central theme. That research note documented how attacks that corrupt the data and metadata agents trust can retain meaningful effectiveness against defenses purpose-built for prompt injection. It recommended data provenance and data-flow isolation as the maturing control category. The Lockdown Mode bypass reaches a parallel conclusion from a different direction. A control

designed around one model of the threat, here outbound network requests from tools, was sidestepped by a path outside that model. Both cases support investment in controls that track where data can flow rather than what instructions look like.

Using Zero Trust to Secure Enterprise Information in LLM Environments [13] supplies the enterprise control set that this note recommends as a compensating layer. That set includes least-privilege access, network controls and micro-segmentation, DLP and CASB deployment, and output validation across both self-hosted and SaaS LLM deployments. Its treatment of SaaS LLM services is directly applicable. The Lockdown Mode incident shows why organizations consuming a SaaS AI product should not assume that vendor-side controls replace their own visibility and egress enforcement.

Securing LLM Backed Systems: Essential Authorization Practices [14] addresses the developer side. It covers design patterns for systems in which LLM output must be treated as untrusted and authorization must be enforced outside the model. Model-generated HTML rendered in a preview pane is exactly such output, and the sandbox and messaging weaknesses in both ChatGPT disclosures are failures to enforce that principle at the rendering layer. Organizations mapping these lessons into auditable controls should use the AI Controls Matrix (AICM) v1.1 [15]. Its Application and Interface Security (AIS) and Threat and Vulnerability Management (TVM) domains are the natural home for requirements on client sandboxing, rendering-surface testing, and vendor vulnerability disclosure. CSA's *Agentic AI Red Teaming Guide* [16] offers test methodology for extending red-team scope from model behavior to the surrounding application and its egress paths.

References

[1] OpenAI. "Introducing Lockdown Mode and Elevated Risk labels in ChatGPT." OpenAI, February 13, 2026 (updated June 2026).

[2] Help Net Security. "ChatGPT gets new security feature to fight prompt injection attacks." Help Net Security, February 16, 2026.

[3] OpenAI. "Lockdown Mode." OpenAI Help Center, accessed November 6, 2026.

[4] Eli Smadja, Check Point Research. "ChatGPT Lockdown Mode Bypass Reveals a Wider Challenge in AI Security." Check Point Blog, November 5, 2026.

[5] Ravie Lakshmanan. "Researchers Bypass ChatGPT Lockdown Mode to Exfiltrate Chats Without User Interaction." The Hacker News, November 5, 2026.

[6] Mukul Goyal, Parameter Labs. "Do the Math: How We Took Over ChatGPT Accounts." Parameter Labs, October 28, 2026.

[7] Simon Willison. "How we took over ChatGPT accounts." Simon Willison's Weblog, October 31, 2026.

[8] Mozilla. "