WHITE PAPER

CYTEX™
AI-Powered Resilience Command Center

ChatGPhish

ChatGPT's Markdown Rendering Becomes a Phishing Vector Through Trusted Web Summaries

hello@cytex.io | https://cytex.io

ChatGPhish - ChatGPT's Markdown Rendering Becomes a Phishing Vector Through Trusted Web Summaries

Technical analysis of the prompt injection and phishing technique that abuses ChatGPT's trust in Markdown content from summarized web pages, surfacing attacker-controlled links, images, and QR codes inside the assistant's trusted interface.

What Happened

Security researchers have disclosed a vulnerability class in OpenAI ChatGPT that converts the assistant's web summarization feature into a phishing delivery surface. The technique, **ChatGPhish**, exploits an implicit trust relationship: when ChatGPT summarizes a web page on the user's behalf, the assistant renders Markdown content from that page directly into its response interface, including live clickable links and auto-fetched images. Any web page a user asks ChatGPT to summarize can therefore inject malicious links, fake security alerts, remote images, and QR codes into a UI the user trusts as authoritative output from the assistant.

The technique is functional against ordinary, unmodified web content. An attacker does not need to compromise the ChatGPT service, the user's account, or the user's device. The attacker only needs to control or contribute content to a web page that the victim will plausibly summarize, and append a small Markdown payload to that page. The next time the victim invokes ChatGPT's summarization on the page, the payload renders inside the assistant's response as if it were ChatGPT's own output.

The disclosure sits inside a broader recognized issue: LLMs cannot reliably distinguish between instructions issued by their user and instructions embedded in retrieved data. This is the issue cataloged as **OWASP LLM01:2025 (Prompt Injection)** in the OWASP Top 10 for LLM applications. ChatGPhish is a concrete operationalization of that abstract risk against a widely used consumer interface.

How the Vulnerability Mechanism Works

The defect lives in the **response renderer**, not in the underlying language model. The renderer's behavior is the exploitable surface:

ChatGPT's response renderer trusts Markdown content from summarized pages. When the assistant summarizes a page, content from that page flows through the model and into the rendered response. Markdown links and image URLs originating in the source page are preserved through the summarization and rendered as functional UI elements in the response.

Image URLs are auto-fetched by the client. When a Markdown image renders in the assistant's response, the client makes an outbound HTTP request to fetch the image. The request originates from the user's browser session, carrying the user's IP address, User-Agent string, and Referer headers. The attacker who controls the image host receives this information passively, with no user interaction required beyond viewing the assistant's response.

Links are surfaced as live, clickable elements. Markdown links in the source page appear in the assistant's response as styled, clickable elements visually indistinguishable from links the assistant would generate on its own.

No origin attribution is displayed. The rendered response does not visibly indicate which portions of the output were generated by the assistant and which were lifted from the source page. The user has no UI signal that a link or image inside ChatGPT's response is attacker-controlled content from a third-party page, not the assistant's own recommendation.

The exploitation pattern is straightforward:

The attacker appends a Markdown payload to any web page the victim might summarize. The payload can include malicious links styled as legitimate references, attacker-hosted images that fire on load, fake security alerts formatted to mimic ChatGPT's own UI conventions, or QR codes served from attacker-controlled storage. When the victim later asks ChatGPT to summarize the page, the payload renders inside the assistant's response. The victim experiences the malicious content as part of the assistant's trusted output.

Attack Scenarios

Malicious clickable links rendered as assistant output. Markdown links inside the summarized content appear as live, styled links in ChatGPT's response. A user who clicks a link rendered inside the assistant's UI does so under the assumption that the assistant generated or vetted the destination. The destination can be a phishing page, a malware download, or a credential capture endpoint.

Fake system-style security alerts. Carefully formatted Markdown can produce content that visually resembles legitimate system alerts ("Your session has expired, click here to reauthenticate"). When rendered inside the assistant's interface, the alert appears as if ChatGPT itself is communicating it. The visual styling of the response matches genuine assistant output, eliminating the visual differentiation a phishing email would normally provide.

QR codes for mobile device pivoting. QR codes embedded as images in the source page render inside the assistant's response and load from attacker-controlled storage (commonly an S3 bucket or equivalent). The QR code attack is operationally significant because it routes the victim onto a mobile device, bypassing desktop-based URL filtering, enterprise web proxy inspection, and endpoint security controls. The mobile device may have looser security posture and lacks the corporate browser controls that would block the equivalent desktop click.

Passive reconnaissance through auto-fetched images. Even without victim interaction beyond viewing the response, every auto-fetched image leaks the victim's IP address, User-Agent, and Referer to the attacker's hosting infrastructure. This is sufficient for fingerprinting, geolocation, and targeting subsequent attack stages.

Output styled identically to legitimate ChatGPT responses. The malicious content inherits ChatGPT's visual styling, font choices, response formatting, and UI conventions. There is no visible distinction between attacker-injected content and content the assistant itself produced. The trust signal the user relies on (the ChatGPT interface) is the same trust signal the attacker leverages.

Why Traditional Security Boundaries Fail

The web security model relies on the **same-origin policy** to constrain what code from one origin can do to content from another. Same-origin policy provides no defense against ChatGPhish because:

The AI assistant executes in the user's authenticated context. The browser treats the assistant's UI as legitimate, user-authorized output. There is no cross-origin request being made by attacker code. The user themselves is asking the assistant to fetch and process the third-party page.

Attacker-controlled content influences rendered output without origin labeling. The boundary between "content the assistant generated" and "content the assistant lifted from the page" is invisible in the rendered response. The browser cannot enforce a security boundary on a distinction that is not made visible in the first place.

The browser becomes a delivery surface, not a defense. Browser security controls, content security policies, and URL filtering are configured against direct user navigation. They are not configured against the indirect path where the user invokes an AI assistant that fetches and renders untrusted content on their behalf.

This pattern is not unique to ChatGPT. Researchers demonstrated the same class of issue against Microsoft Copilot in March 2026 through **cross-prompt injection (XPIA)**, where an attacker-controlled email containing crafted instructions influenced Copilot's output when the assistant summarized the email. The underlying defect is the same: LLM-mediated interfaces cannot distinguish instructions in the user's request from instructions in the retrieved content the assistant is processing on the user's behalf.

Indicators and Detection Signals

ChatGPhish does not produce traditional malware IOCs. The exploitation primitive uses ordinary Markdown features and legitimate HTTP image fetches. Detection focuses on behavioral patterns:

Browser and network signals:

- Outbound image fetch requests originating from the ChatGPT web interface to domains unrelated to the user's session, particularly to URL-shortened endpoints, S3 buckets without prior interaction history, or recently registered domains
- HTTP requests with the ChatGPT origin in the Referer header reaching destinations outside the OpenAI-controlled infrastructure

- QR code image fetches from external sources rendered inside ChatGPT sessions
- Patterns of outbound requests immediately following web summarization invocations

User-side signals:

- ChatGPT responses containing clickable links to domains the user has not historically interacted with
- ChatGPT responses containing content that mimics system alerts, login prompts, or security warnings
- ChatGPT responses where the linked or imaged content does not align with the source page's purpose

Enterprise environment signals:

- AI browser activity logs showing unexpected image fetch requests to unfamiliar or shortened-URL endpoints
- Mobile devices scanning QR codes immediately following ChatGPT session activity on a corresponding desktop
- Phishing reports from users that reference content seen inside an AI assistant interface rather than in email

Mitigations

Avoid AI summarization on pages with user-generated or untrusted content. Public Reddit threads, third-party GitHub READMEs, comment-enabled blogs, public forums, and any page where an attacker can contribute content are all high-risk surfaces. The summarization feature is safest on pages whose content is fully controlled by a trusted publisher.

Restrict AI browser permissions to the minimum necessary. Where the AI assistant offers configurable permissions for link interaction, image loading, and external resource access, default to the most restrictive setting. Require explicit human approval before any link inside a summarized response is followed.

Treat all clickable links, images, and alerts inside AI summaries as potentially attacker-controlled until origin attribution is clearly displayed. This is a user behavior change rather than a technical control. Until AI assistants surface clear visual distinction between assistant-generated content and content lifted from a source page, users should not extend assistant-level trust to any UI element appearing inside a summarized response.

Deploy semantic input and output filtering on enterprise AI-integrated surfaces. Where AI assistants are integrated into enterprise environments through API access, browser extensions, or workplace productivity tools, apply filtering on both the content being submitted to the assistant

and the content being rendered back. Prompt injection patterns, suspicious URL classes, and image fetch behaviors are all candidates for filtering and anomaly detection.

Monitor AI browser activity logs for unexpected outbound fetches. Centralize logging of AI assistant interactions where possible and correlate outbound network activity from AI-integrated browsers with the timing of summarization requests. Fetches to unfamiliar or URL-shortened endpoints during summarization sessions are a strong behavioral indicator.

Apply standard phishing-resistant authentication. Where the attack vector terminates in a credential capture page, FIDO2 hardware keys, certificate-based authentication, and other phishing-resistant mechanisms defeat the credential theft step even when the user reaches the phishing destination.

Educate users on the assistant interface trust model. Most users have not yet calibrated the trust assumption appropriate to AI assistant output. The intuitive trust signal ("this came from ChatGPT, so it must be safe") is not aligned with the actual security model ("this came from ChatGPT's rendering of a page someone else wrote"). Closing that calibration gap is a training and communication exercise that should accompany any enterprise rollout of AI assistant features.

What This Disclosure Reveals

The structural lesson of ChatGPhish extends beyond the specific Markdown rendering defect. **AI assistants that fetch, process, and render third-party content on the user's behalf inherit the trust of the user while exposing the user to the content of the third party.** The boundary between "assistant output" and "third-party content the assistant happened to display" must be made explicit in the interface, or the trust the user places in the assistant becomes a vector the attacker can hijack.

Until source separation between retrieved web content and rendered assistant output is enforced at the UI layer, browser-integrated AI summarization remains an adversarial surface. The vulnerability class is not specific to ChatGPT, not specific to Markdown, and not specific to web summarization. It applies to any LLM-mediated interface where retrieved content is rendered with the same visual authority as the assistant's own output.