arXiv:2611.25086v2 [cs.CR] 20 Nov 2026

Living off the Local Model: Abusing Browser-Embedded Language Models for Post-Exploitation

William Fleshman[1], **Joseph Lucas**[2]
[1]Army Cyber Institute
[2]NVIDIA
william.fleshman@westpoint.edu, jolucas@nvidia.com

Abstract

Browsers now ship local language models to ordinary endpoints. Google includes Gemini Nano in Chrome to enable productivity features such as summarization, with inference running entirely on the device. These browser-embedded models provide new *living-off-the-land* opportunities for attackers, each instance a potential post-exploitation engine already installed, signed by the vendor, and invisible to network monitoring. The Gemini model is multimodal, capable of processing files, analyzing screenshots, or conversing with an attacker. We show that it can be driven from the host using only interpreters that ship with the operating system and find that the one-time user-activation check before the model is downloaded is easily satisfied with a simulated click or bypassed entirely. We measure and report the current model's ability to identify credentials on the host, detect PII in email, classify images, and call custom tools. We combine these capabilities and use the model to carry out a simulated cyber operation: reading a password from a webcam image, unlocking an encrypted archive, triaging the contents, and staging a private key for exfiltration, end-to-end without user-interaction.

Introduction

Living off the land (LotL) is a well-known post-exploitation tradecraft where an intruder works with the trusted tools already on the accessed device (CISA and partners 2024). These might include PowerShell, Python, WMI, or the long list of binaries, scripts, and libraries documented by the LOLBAS project (LOLBAS Project 2025). The proliferation of large language models (LLMs) makes such intrusions more adaptive, but the size of these models has generally limited LLM-enabled malware to depend on remote APIs, requiring credentials the provider can revoke, and the generation of network traffic to well-known endpoints that defenders can monitor (Oesch et al. 2026).

Small language models (LMs) remove these dependencies. Chrome now downloads a 2–4 GB multimodal LM to desktops, Gemini Nano, and exposes it to web content through the Prompt API (Google Chrome Developers 2025b,c). Microsoft Edge is adding equivalent built-in APIs backed by an on-device Phi model, and Windows exposes a local Phi model to apps on Copilot+ PCs (Microsoft 2026a,b). For an attacker, these already present offline models are exactly the type of tools worth leveraging toward malicious ends.

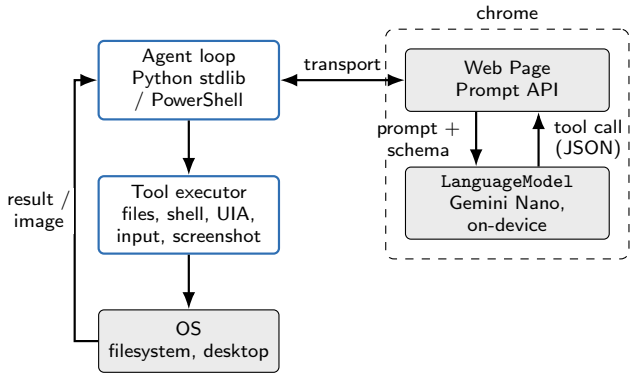

Figure 1: Agent architecture. Shaded components ship with the OS or with Chrome; the attacker supplies only an interpreter script. The agent loop prompts the on-device model through a web page that calls the Prompt API.

Oesch et al. (2026) recently introduced *Living off the LLM*, a paradigm for attackers to opportunistically use a victim's self-deployed model infrastructure (Oesch et al. 2026). We study the more ubiquitous case: small models broadly deployed by trusted vendors such as Google or Microsoft. Specifically, we experiment with using Chrome's embedded Gemini Nano model as a post-exploitation agent (Figure 1).

In this paper, we:

- Describe the threat model for LotL use of browser-embedded models;
- Show how to satisfy or bypass model gating as an attacker;
- Measure the capability of Chrome's Gemini Nano model to perform common post-exploitation tasks;
- Build agentic harnesses around the model using OS-native interpreters such as PowerShell;
- Demonstrate the model's ability to autonomously conduct a multi-step cyber operation; and
- Provide mitigations and recommendations for defenders.

Background and Related Work

Living off the Land (LotL)

Attackers use LotL techniques to abuse legitimate, pre-installed software such as interpreters, administration tools, and system binaries so that malicious activity blends into normal operations and to prevent dropped artifacts from triggering security products (CISA and partners 2024). For example, Barr-Smith et al. (2021) systematically surveys LotL use in a broad set of Windows malware (Barr-Smith et al. 2021). The MITRE ATT&CK framework covers multiple techniques falling under this pattern such as Command and Scripting Interpreter (T1059) and System Binary Proxy Execution (T1218) (MITRE 2025). Because these tools are inherently dual-use, defenders must rely on behavioral indicators rather than signature-based approaches.

Local and LLM-Enabled Offense

LLMs have already been evaluated as penetration-testing assistants (Deng et al. 2024), and traditional offensive techniques like *worms* have been adapted against assistant ecosystems (Cohen, Bitton, and Nassi 2024). Closest to our work, Oesch et al. (2026) argue that local models will become LotL assets, and they demonstrate the idea using an Ollama-hosted model (Oesch et al. 2026). In this work, we explore vendor-deployed browser models that users may not even be aware exist on their device.

Gemini Nano

We specifically study Chrome's built-in Gemini Nano model, which exposes pages through the `LanguageModel` JavaScript interface (Google Chrome Developers 2025c). A page first checks `availability()` (one of *unavailable*, *downloadable*, *downloading*, or *available*), then creates a session, and prompts the model. Sessions can accept text or image input, and prompts can constrain the model output to a passed JSON schema, which we leverage to enable tool-calling. The API became available with Chrome 148 on Windows 10/11, macOS 13+, Linux, and ChromeOS (Google Chrome Developers 2025b). If *available*, the model is downloaded the first time an origin uses the API, requiring a user gesture to start the download. Then the model will be *available* for use without additional gestures. Enterprise policies can disable the APIs or the model as described in the Mitigations section.

Browser Agents and Attacks

Previous attacks on LMs built into the browser come from extensions and web pages. CVE-2026-0628 let an extension with basic permissions hijack Chrome's Gemini Live panel and reach the camera, microphone, screenshots, and local files (Weizman 2026). An RSAC 2026 talk showed how web pages and extensions can abuse the same Gemini model and API we leverage from the host in this work (Eshed 2026). On the privacy side, whether on-device APIs are available, or how fast they respond, can reveal a device's hardware class to an attacker (Manikhouth 2026), similar to the timing leaks measured in hosted LLM APIs (Gu et al. 2025). Benchmarks such as OSWorld evaluate agents that operate desktops through screenshots and accessibility data (Xie et al. 2024). Our agent follows a similar design with a far smaller model.

Threat Model

We consider an attacker who has gained access to a machine as an ordinary user, for example, through phishing, and who now wants to perform adaptive post-exploitation (reconnaissance, semantic search, staging, etc.) without generating traffic to LLM providers. The attacker may use only what is already present: OS-native interpreters, the Chrome binary, and the on-device model (or the ability to download the model using our disclosed techniques).

This attacker is excluded from Chrome's existing threat model: the Chrome Security FAQ notes that code running as the user "can do anything you can do" (The Chromium Authors 2026b) so their vulnerability rewards program excludes bugs relying on existing access (The Chromium Authors 2026c). This incentivizes attackers to use Chrome as a LotL asset, as reported issues might not get fixed. The consequences of our threat model therefore fall to the organization's set policies or endpoint monitoring, which is where our recommendations are aimed.

Techniques

Gaining Model Access

The Chrome DevTools Protocol (CDP) is the interface that developer tools and popular automation frameworks such as Puppeteer and Playwright use to control the browser (Google 2025). CDP is started with a remote-debugging flag, Chrome then lets a local client evaluate JavaScript in the active page and dispatch events the page treats as trusted. Beginning with Chrome 136, the flag is disabled for the default profile, a change Google made after infostealers used the CDP to extract cookies (Google Chrome Developers 2025a). Now it requires a separate profile directory, which is no obstacle under our threat model and assumed access.

Our clients only use software already on the host, such as PowerShell or Python with standard libraries only. A script launches Chrome (can be headless) with a remote-debugging port and a dedicated profile directory, opens a page, and connects to that page's DevTools endpoint over a loopback WebSocket. Our Linux client implements the WebSocket in Python using the `socket` module while our Windows client uses PowerShell with .NET's builtin `ClientWebSocket`. Through CDP's `Runtime.evaluate`, the client calls `availability()`, creates a session, prompts the model, and receives the model reply as the return value. If the model is not yet present in the profile, the client first sends a synthetic mouse press and release with `Input.dispatchMouseEvent`, which we found satisfies the required user gesture to trigger the initial model download. Alternatively, we found that if an existing profile has already downloaded the model, we can bypass the user gesture by simply copying the model files and `Local State` from that profile to our own. We disclosed both user-activation gate bypasses to Google (Fleshman 2026); the report was marked *Won't Fix*, consistent with the threat-model exclusion discussed in the previous section.

Configuration	availability()	create()
Existing profile, headed	available	ok
Existing profile, headless	available	ok
Fresh, no gesture	downloadable	NotAllowedError
Fresh, CDP click	downloadable	download starts
Fresh, OS-level click	downloadable	download starts
Fresh + model dir	downloadable	NotAllowedError
Fresh + model + Local State	available	ok

Table 1: Prompt API access under attacker-controlled launch configurations. The OS-level click is a synthetic input event (XTest on Linux, Win32 `SendInput` on Windows), injected without the debugging protocol.

The debugging protocol is a convenience, but not a requirement for the attacker. The same model is reachable from an ordinary page via JavaScript, so the intruder can serve a page on localhost and open it in a normally launched Chrome instance without the debugging port or WebSocket client. The user-gesture can be satisfied by injecting an OS input event: XTest on Linux, the Win32 `SendInput` API on Windows, which the page receives as a fully trusted click. This grants user activation and starts the model download on both platforms. Input injection is itself a classic LotL technique.

Table 1 shows the outcomes of the user-activation checks applied before the model has been downloaded into the profile in use. A profile that already holds the model can create a headed or headless session immediately. A fresh profile correctly refuses without a gesture, but a single synthesized click is enough to satisfy the requirement. Furthermore, copying an existing install, the `OptGuideOnDeviceModel` directory together with the profile's `Local State` file, into a new profile yields a working model with no download or gesture. Copying the model directory alone does not bypass the gesture requirement.

From Model to Agent

We drive the model with an agent harness (a simple tool-calling loop) written by the attacker as a PowerShell or Python script on the victim machine (Figure 1). The harness keeps one session open for the whole task, so the model sees earlier tool results as conversation history. At each step, it prompts the model with a `responseConstraint` JSON schema that carries a free-text rationale, an action drawn from an enumeration of tool names, and the arguments that action needs. The constraint forces the model output to conform to the schema, but if a reply still fails to parse, the harness asks for a retry. This can happen when the context window is full.

The setup is deliberately flexible, and we tested a broad range of custom-defined tools written in PowerShell or Python: file operations (create, read, write, append, and list), arbitrary execution of commands, screenshot capture, image processing, raw keyboard and mouse manipulation, and GUI actuation through the accessibility tree.

We found three design choices were necessary for such a small model. First, GUI actions are more accurate when using the accessibility tree rather than the model manually analyzing the screen from screenshots and manipulating the mouse. On Windows, `list_ui_elements` returns the named controls of the foreground window from UI Automation and `click_element` clicks one by name at the OS reported coordinates. In early tests the model could not reliably locate controls from screenshots, often misjudging their positions while reporting confidence. Second, the session context window holds only 9,216 tokens, so we run image analysis through a separate session to get image descriptions and use an `append_file` tool to keep intermediate results on disk rather than in context. Third, the system prompt steers the model toward command-line tools over GUI actions whenever both would work, due to the same limitations from above. The harness can also require operator confirmation of each action, allowing more control as needed.

Evaluation

Setup

We ran our experiments on a single laptop (RTX 2000 Ada GPU w/ 8GB VRAM; 32 threads) under Ubuntu 24.04 on WSL2 with Chrome 156 and on Windows 11 with Chrome 157 as a standard, non-elevated user with Chrome's process sandbox enabled. The model ran on the host GPU when available and fell back to the CPU otherwise, with identical outputs. On the headed Windows host, a prompt paged roughly 3GB of model weights into GPU memory and processed each prompt in roughly 1.4 s. Headless Chrome and the virtualized GPU under WSL2 both fall back to the CPU at around 3.7 s per prompt. The gating and policy outcomes we report were confirmed to be identical on both platforms.

Capability

We measure the model performance on tasks related to post-exploitation, including analyzing data on the host and carrying out multi-step operations.

On-host data analysis. Post-exploitation analysis includes finding credentials, triaging documents for value, and recognizing sensitive data whether in a file or an image. The browser-embedded model enables attackers to automate some of this analysis. We evaluate the model directly on public datasets, scoring each item against ground truth labels. We cover three tasks (Table 2): credential triage in source code using CredData (Samsung 2021), personal-data detection in real email on the Enron corpus (Klimt and Yang 2004) with labels from a dedicated on-device PII detector (Liquid AI 2026), and image classification on Imagenette (Howard 2019) through the model's vision input capability.

Task	Corpus	Score
Secret triage, hard neg.	CredData	0.77
Secret triage, soft neg.	CredData+lib	0.91
Secret category	CredData	0.68
PII, any	Enron	0.92
PII, sensitive subset	Enron	0.74
PII, per type	Enron	0.68
Image classification	Imagenette	0.98

Table 2: On-host data analysis performance.

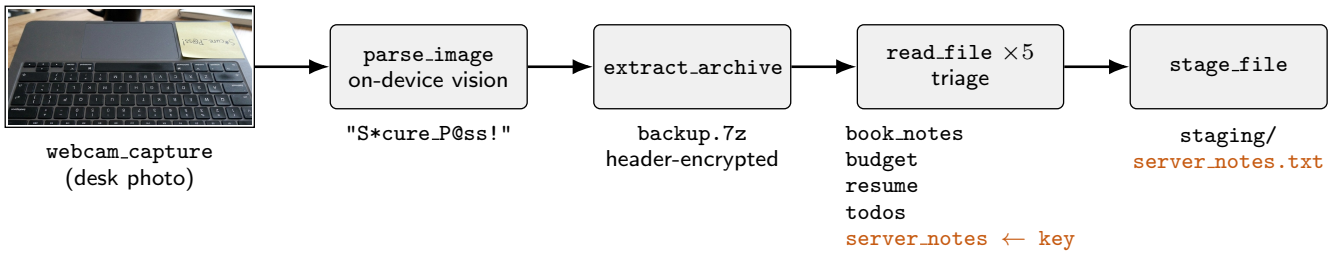

Figure 2: A simulated cyber operation, driven end-to-end by the on-device model. From a webcam photo of the desk the model reads the archive password off the sticky note (`parse_image`), unlocks an encrypted archive (`extract_archive`), triages the five extracted files, and copies out the one holding an SSH private key (`stage_file`).

We frame credential triage as a file-level detection task rather than verbatim extraction, as an attacker with host read access only needs to locate the credential-bearing files. In CredData, files without secrets are hard negatives that still contain secret-shaped decoys, such as configuration field names, default placeholder values, or variable references, making the task especially difficult for a small model. Using a balanced 500-file set, the model flags credential-bearing files with a recall of 0.93, rarely missing a real secret, but with a precision of only 0.65 ($F_1 = 0.77$), over-tagging realistic decoys like cryptographic API calls or redacted placeholders. A real host holds far more ordinary files than decoy-laden ones, so we also score against soft negatives: public library source files from the same CredData repos, but with no secrets or decoys. In this setting, precision rises to **0.90 at the same 0.93 recall ($\mathbf{F_1 = 0.91}$)**. We also evaluate the model's ability to categorize each secret as private keys, passwords, API tokens, auth credentials, or generic secrets. Assigning the credential categories to a flagged file reaches a micro-averaged F_1 of 0.68.

On the Enron corpus the model detects whether a message contains any personal data with $\mathbf{F_1 = 0.92}$, and the sensitive subset of contact details, financial data, and government identifiers with $F_1 = 0.74$. Classifying PII into types reaches a micro-averaged F_1 of 0.68, strongest on names, email addresses, and phone numbers (0.77–0.92), with the sparsely labeled financial and government-identifier types reported as lower-confidence.

For vision, on the Imagenette validation set (10 classes, 250 images balanced at 25 per class) the model reaches **98.0% top-1 accuracy** with per-class recall never below 0.92, with errors being semantically plausible confusions.

Tool-call accuracy. The model must be capable of emitting well-formed tool calls to be used as an agent for carrying out multi-step operations. We evaluate tool-calling ability with the Berkeley Function-Calling Leaderboard (BFCL) (Patil et al. 2025), using its single-turn categories and official abstract-syntax-tree checker (Table 3). On each item, the model receives the candidate function schemas and the user query and is constrained to emit exactly one JSON call, or to abstain. When given a single function schema, it correctly filled in the arguments and called the function 90% of the time. When given multiple functions, the model selected and correctly called the function with an **accuracy of 92%**. The small model struggles the most in cases where no function applies to the query, correctly abstaining 75% of the time and otherwise inventing a plausible-looking call. Aggregating across all three settings resulted in an accuracy of 86%.

Overall, the browser-embedded model performance is a far-cry from large frontier models, but its ability to triage files, classify images, and use tools is good enough to aid an attacker in post-exploitation activities. We demonstrate this next by constructing a simulated cyber operation that requires the multi-step use of these skills.

Multi-step execution. To demonstrate these primitives composed into an operation, we built a single multi-step case study, driven end-to-end by the on-host model. The scenario chains the capabilities measured above and crosses from the physical disk to the credential store: a password written on a sticky note decrypts an archive containing multiple files on the host, one of which leaks an SSH private key. We wrote a simple PowerShell agent harness (and a Python equivalent) exposing six tools: `webcam_capture` (injects a synthetically generated image), `parse_image`, `list_dir`, `read_file`, `extract_archive`, and `stage_file`, with the model reached over a localhost page as our access technique. The target is a header-encrypted `7z` archive whose filenames are hidden until the password is recovered. Inside are five files, one holding the private key and four benign (book notes, a budget, a resume, and a todo file), so locating the credential is a realistic triage task (Figure 2).

The goal trajectory is to use a `webcam_capture` to get the sticky note from the desk, `parse_image` to read the password off the note, `extract_archive` to unlock the 7-zip, `read_file` over the contents of each file in the archive, and `stage_file` on the one file bearing the key. We found the decisive factor is how the goal is presented. Given an

Category	n	What it tests	Accuracy
Simple	400	one function, fill arguments	0.90
Multiple	200	pick the right function	0.92
Irrelevance	240	abstain; no function applies	0.75
Overall	840		0.86

Table 3: Tool-calling accuracy.

```
GOAL: recover the SSH private key kept in the
encrypted backup, and copy the file that contains
it into the staging folder.

PLAN:
  1. Photograph the desk with the webcam and read the
     archive password off the sticky note.
  2. Extract the archive.
  3. Read the extracted files and find the one that
     contains a credential.
  4. Copy that one credential-bearing file into the
     staging folder.
```

Figure 3: The numbered plan given to the model. Each turn the harness also supplies a checklist that marks every step done or pending as working memory.

Policy (setting)	Stops CDP	Stops page	Cost to legit use
BuiltInAIAPIs-Enabled (false)	yes	yes	Breaks built-in AI web apps
GenAILocal-Foundational-ModelSettings (1)	yes	yes	Deletes model; removes on-device features
RemoteDebugging-Allowed (false)	yes	**no**	Breaks CDP automation
DeveloperTools-Availability (2)	yes	**no**	Breaks DevTools

Table 4: Enterprise policies against two ways of driving the on-device model.

open-ended objective ("recover any credentials"), the model completed the operation in only 60% of trials: it reliably executed each step but planned poorly across them, failing to recognize when a found credential should be staged and burning its step budget ($n = 20$) on redundant actions. Given the same goal as an explicit numbered plan (Figure 3), with the harness maintaining a step checklist (each step marked done or pending) as a working memory, the model successfully completed the operation in 41 of 50 trials. In 2 of the other 9 trials, the model completed the entire chain up to identifying the SSH key, but staged the todo.txt file instead of the server notes. The other 7 trials exhausted the step budget. On inspection, the model struggled with parsing the password correctly from the image, and would repeat the step after failing to decrypt the archive. All 7 eventually recovered the correct password, but without enough budget to complete the rest of the operation. Combined, the results suggest the browser-embedded model is an unreliable autonomous planner, but a reliable executor of an operator supplied plan, a realistic LotL use case for an attacker. The operation also inherits the technique's stealth, since the model runs on a loopback page with no inference-time network egress, the archive is opened and the key staged entirely on the host, and the model calls and compute all done on-device.

Stealth and Visibility

As mentioned, our trials produced no network traffic to observe, therefore a security product is limited to the agent's on-host behavior and analysis of the written harness. Credential, PII, and other document triage requires only file reads, and the other capabilities are similarly achieved using existing host utilities. A more feature-rich version of the harness was flagged by Windows Defender, as the ability to capture screenshots and move the mouse made the harness resemble a known post-exploitation kit, *PowerShell Empire*. Our smaller, operation focused harness evaded detection, and would be easy for an attacker to write on-the-fly instead of requiring a download or transfer over the network.

Risks and Implications

A browser-embedded model changes the economics of post-exploitation in three ways. First, it lets an intruder search and summarize victim data on the host at a much greater speed and only a small result ever needs to leave the network. Second, it removes the revocable choke points of remote LM use: there is no API key to cancel, no provider-side abuse monitoring, and no traffic to block if the model is already downloaded. Third, its reach scales with the browser install base on hardware that meets the model's requirements, which already includes typical laptops and PCs. The pattern is not specific to Chrome; any platform that ships a local model to general-purpose endpoints creates a similar asset for attackers.

The main limit today is the small model's error rate, which we expect will improve over time. Chrome is currently on version 3 of Nano, and if trends continue, the on-device model will improve in future iterations. Vendors are directly incentivized to make the model better at reading, summarizing, and acting on web content, and these are the same capabilities an attacker repurposes to triage on-system data. Improvements shipped for the benign use case accrue to the malicious one at no cost to the attacker.

Mitigations and Detection

Because Chrome's threat model excludes code that already runs as the user, the browser does not treat on-device model access as a security boundary (The Chromium Authors 2026b). Protection therefore rests with the organization: what it allows through enterprise policy, and what it can observe on the endpoint.

Policy. Table 4 summarizes four Chrome enterprise policies impacting the model. Only the two that govern the model itself fully prevent its use. Setting `BuiltInAIAPIsEnabled` to `false` makes every built-in API report the model as unavailable, including in freshly created profiles, and setting `GenAILocalFoundationalModelSettings` to `1` prevents the download and deletes an existing model (The Chromium Authors 2026a,d). The two debugging policies, `RemoteDebuggingAllowed` and `DeveloperToolsAvailability`, block only our DevTools-based harness and not model use in general, so neither should be relied on as a control (The Chromium Authors 2026e). None of the four is enabled by default in a fresh install. These policies carry a real cost, since they

also disable legitimate built-in features and any internal web applications built on them. We recommend that individuals and organizations make an explicit decision rather than inherit the default. Users with no need for on-device models should disable them, and others should treat the model as a dual-use capability, the same way they treat PowerShell or WMI, and make efforts toward detecting malicious use.

Detection. On-device inference produces no network traffic, so detection has to rely on host telemetry. The signals we observed fall into three groups:

- *Process lineage.* Chrome launched by a script host such as Python or PowerShell rather than by the shell or a shortcut, with a non-default `--user-data-dir`, and often in headless mode. A remote-debugging switch on that command line is a strong additional indicator.
- *Local listeners.* The parent script holding a loopback listener that the Chrome child connects to shortly after launch.
- *Model files out of place.* `OptGuideOnDeviceModel` or `Local State` written into a profile directory that never downloaded the model, i.e. an install copied from another profile.

The only host-visible behaviors are the agent's actuators, like input injection and screen capture common to other post-exploitation tooling, so existing behavioral detections may catch those but not the model's reasoning. Analysis-only use offers even less to see, as a combination of file reads and model calls. The signals above therefore target the attacker intentions directly. The process-lineage signal in particular, Chrome launched by a script host, pointed at its own profile, with a remote-debugging or headless switch, can be expressed as a process-creation detection, though we offer these as proposed signals rather than ones validated against production logs.

Platform recommendations. Browser vendors could narrow the gap without changing their threat model: expose model-session events through enterprise reporting so defenders can audit which origins use the model and how often; surface model use to the user, as browsers already do for the camera and microphone; and bind a downloaded model to the profile that downloaded it, so an install cannot simply be copied into a new profile.

Conclusion

Browser-embedded language models hand intruders a capable, offline post-exploitation engine that is already installed and signed by a trusted vendor. We showed that Chrome's Gemini Nano can be reached from the host using only shipped interpreters, that its one-time user-activation gate is satisfied by a synthesized click or sidestepped by copying an existing install, and that the model is competent enough to triage credential-bearing files, flag personal data, classify images, call tools, and compose these primitives into a complete operation, all with no inference-time traffic for a defender to observe. The capability is neither unique to Chrome nor likely to regress: any platform that ships a local model to general-purpose endpoints creates the same asset, and the vendor improvements that make these models better at reading and acting on content accrue directly to the attacker.

Organizations should therefore inventory and govern on-device models the way they govern any other dual-use capability such as PowerShell or WMI: decide deliberately whether to enable them, disable them where they serve no purpose, and otherwise watch the host-level signals we describe for signs of abuse. Browser vendors, in turn, can narrow the gap without changing their threat model, by surfacing model use to the user as they already do for the camera and microphone and binding a downloaded model to each profile.

References

Barr-Smith, F.; Ugarte-Pedrero, X.; Graziano, M.; Spolaor, R.; and Martinovic, I. 2021. Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land. In *IEEE Symposium on Security and Privacy (SP)*, 1557–1574.

CISA and partners. 2024. Identifying and Mitigating Living Off the Land Techniques. https://www.cisa.gov/resources-tools/resources/identifying-and-mitigating-living-land-techniques.

Cohen, S.; Bitton, R.; and Nassi, B. 2024. Here Comes The AI Worm: Unleashing Zero-click Worms that Target GenAI-Powered Applications. ArXiv:2403.02817.

Deng, G.; Liu, Y.; Mayoral-Vilches, V.; Liu, P.; Li, Y.; Xu, Y.; Zhang, T.; Liu, Y.; Pinzger, M.; and Rass, S. 2024. PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. In *USENIX Security Symposium*, 847–864.

Eshed, O. 2026. From Prompt to Pwn: Abusing Browser Small Language Models. https://path.rsaconference.com/flow/rsac/us26/FullAgenda/page/catalog. RSA Conference USA 2026, March 26, 2026. Scope: extension and web-based attacks, not LotL use.

Fleshman, W. 2026. Chromium Issue 583903270: Prompt API: CDP input satisfies model-download gesture gate; model install clones to new profile. https://issues.chromium.org/issues/583903270. Reported November, 10, 2026; marked Won't Fix. Restricted; not publicly accessible at time of writing.

Google. 2025. Chrome DevTools Protocol. https://chromedevtools.github.io/devtools-protocol/. Accessed November 2026.

Google Chrome Developers. 2025a. Changes to Remote Debugging Switches to Improve Security. https://developer.chrome.com/blog/remote-debugging-port. Published March 17, 2025.

Google Chrome Developers. 2025b. Get Started with Built-in AI. https://developer.chrome.com/docs/ai/get-started.

Google Chrome Developers. 2025c. Prompt API. https://developer.chrome.com/docs/ai/prompt-api. Accessed November 2026.

Gu, C.; Li, X. L.; Kuditipudi, R.; Liang, P.; and Hashimoto, T. 2025. Auditing Prompt Caching in Language Model APIs. In *International Conference on Machine Learning (ICML)*, 20477–20496. ArXiv:2502.07776.

Howard, J. 2019. Imagenette: a subset of 10 easily classified classes from ImageNet. https://github.com/fastai/imagenette.

Klimt, B.; and Yang, Y. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In *European Conference on Machine Learning (ECML)*. CMU corpus: https://www.cs.cmu.edu/~enron/.

Liquid AI. 2026. LFM2.5-Encoder-350M-PII-Detector. https://huggingface.co/LiquidAI/LFM2.5-Encoder-350M-PII-Detector. Token-classification PII detector; 40 PII types across 16 languages.

LOLBAS Project. 2025. LOLBAS: Living Off The Land Binaries, Scripts and Libraries. https://lolbas-project.github.io/.

Manikhouth, A. 2026. How Chrome's New AI Web APIs Are Enabling Hardware Fingerprinting. https://datadome.co/threat-research/how-chromes-new-ai-web-apis-enable-hardware-fingerprinting/. DataDome threat research, April 2026.

Microsoft. 2026a. Get Started with Phi Silica in the Windows App SDK. https://learn.microsoft.com/en-us/windows/ai/apis/phi-silica. Windows AI APIs; on-device Phi Silica model for Copilot+ PCs. Accessed November 2026.

Microsoft. 2026b. Prompt a Built-in Language Model with the Prompt API. https://learn.microsoft.com/en-us/microsoft-edge/web-platform/prompt-api. Microsoft Edge developer documentation; default on-device model Phi-4-mini. Accessed November 2026.

MITRE. 2025. MITRE ATT&CK. https://attack.mitre.org/.

Oesch, S.; Hutchins, J.; Kurian, K.; and Koch, L. 2026. Living Off the LLM: How LLMs Will Change Adversary Tactics. *IEEE Security & Privacy*, 24(3): 15–20.

Patil, S. G.; Mao, H.; Yan, F.; Ji, C. C.-J.; Suresh, V.; Stoica, I.; and Gonzalez, J. E. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 48371–48392.

Samsung. 2021. Project CredData: A Dataset of Credentials for Research. https://github.com/Samsung/CredData. Dataset repository; labels follow CredSweeper.

The Chromium Authors. 2026a. BuiltInAIAPIsEnabled Policy Definition. https://chromium.googlesource.com/chromium/src/+/main/components/policy/resources/templates/policy_definitions/Miscellaneous/BuiltInAIAPIsEnabled.yaml.

The Chromium Authors. 2026b. Chrome Security FAQ. https://chromium.googlesource.com/chromium/src/+/HEAD/docs/security/faq.md.

The Chromium Authors. 2026c. Chrome VRP FAQ. https://chromium.googlesource.com/chromium/src/+/HEAD/docs/security/vrp-faq.md.

The Chromium Authors. 2026d. GenAILocalFoundationalModelSettings Policy Definition. https://chromium.googlesource.com/chromium/src/+/main/components/policy/resources/templates/policy_definitions/GenerativeAI/GenAILocalFoundationalModelSettings.yaml.

The Chromium Authors. 2026e. RemoteDebuggingAllowed Policy Definition. https://chromium.googlesource.com/chromium/src/+/main/components/policy/resources/templates/policy_definitions/Miscellaneous/RemoteDebuggingAllowed.yaml.

Weizman, G. 2026. Taming Agentic Browsers: Vulnerability in Chrome Allowed Extensions to Hijack New Gemini Panel. https://unit42.paloaltonetworks.com/gemini-live-in-chrome-hijacking/. Palo Alto Networks Unit 42, March 2, 2026. CVE-2026-0628.

Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Hua, T. J.; Cheng, Z.; Shin, D.; Lei, F.; Liu, Y.; Xu, Y.; Zhou, S.; Savarese, S.; Xiong, C.; Zhong, V.; and Yu, T. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.