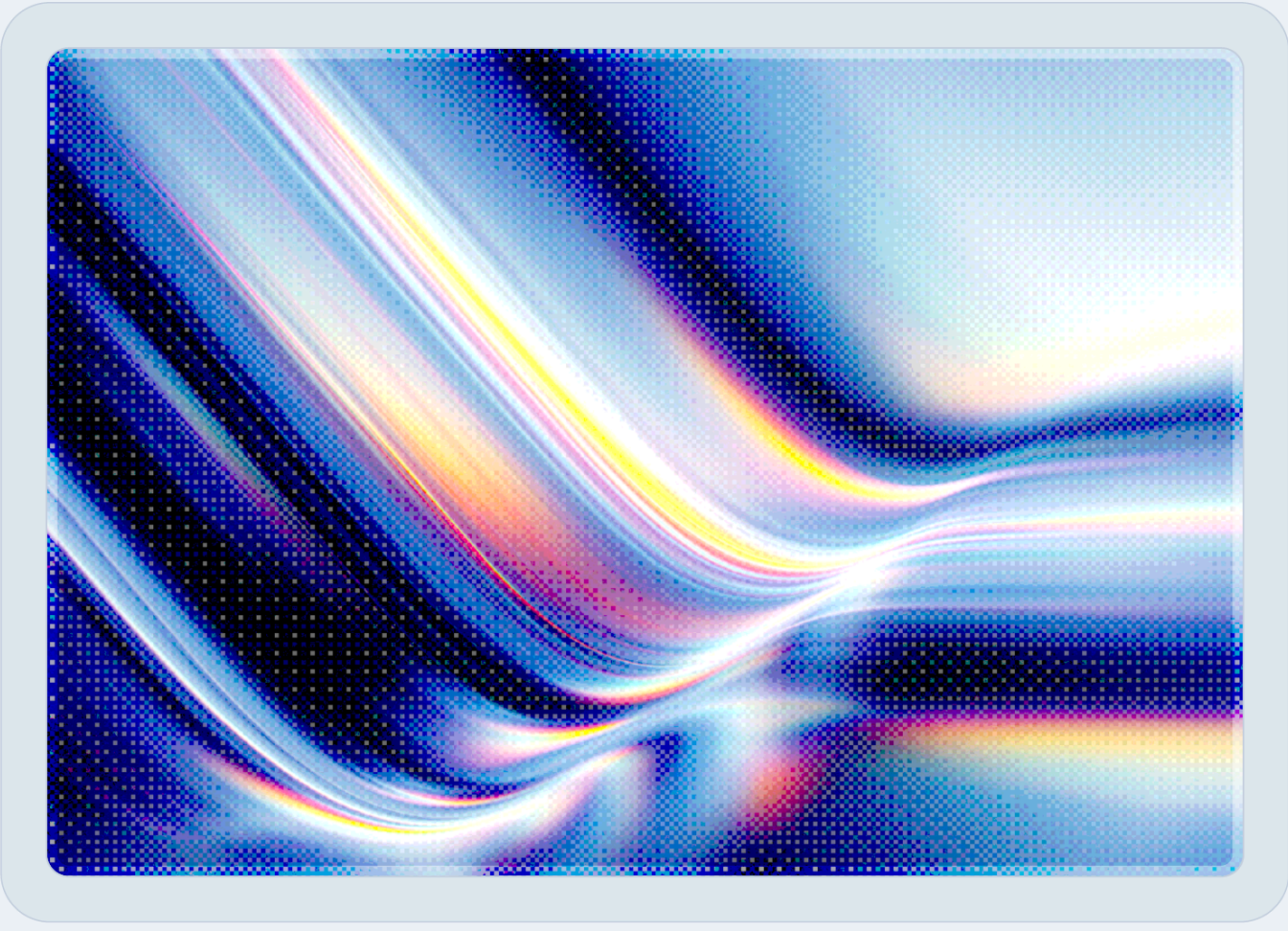
CSAI Foundation | Cloud Security Alliance

CDP Session Hijacking Bypasses Device-Bound Credentials

How Living-off-the-DevTools-Protocol Defeats Chrome's Hardware-Bound Cookies

2026-08-17

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Security researchers at SpecterOps published a working post-exploitation technique that enables the Chrome DevTools Protocol (CDP) inside an already-running `chrome.exe` or `msedge.exe` process, giving an attacker with existing code execution the same API access to a victim's authenticated browser session that the browser's own developer tools have, without touching the disk-based cookie store [1][2]. The technique, released publicly on August 13, 2026 as the open-source tools CDP-Enable-BOF and CDP Toolkit, builds directly on Cedric Van Bockhaven's research on "living off the DevTools Protocol" and on work by the researcher known as DeathFlamingo, who documented CDP injection into a running Edge browser in December 2025 [1][2]. Because the attacker operates through the browser's own UI thread rather than replaying a stolen credential from a separate device, the method sidesteps Google's Device Bound Session Credentials (DBSC), a hardware-key-backed protection that Google made generally available for Windows users starting with Chrome 146 and designed to make a stolen or replayed session cookie unusable from anywhere but the device it was issued to [3][4]. SpecterOps was explicit that it did not extract the DBSC private key itself; instead, the technique borrows the already-authenticated browser to issue requests, including calls to the CDP `Storage.getCookies` endpoint, that never require the key to leave the device [2]. The disclosure follows a pattern Google itself has flagged: after Chrome's April 2025 rollout of App-Bound Encryption made disk-based cookie theft harder, Google reported an increase in attackers pivoting to Chrome's remote-debugging interface instead [5][6]. The technique requires an attacker to already have code execution on the endpoint — it is a post-exploitation and lateral-movement tool, not a remote browser exploit — but, in CSA's assessment, it materially raises the value of any foothold on a machine with an open, authenticated enterprise session.

Background

The Chrome DevTools Protocol is the JSON-RPC interface that Chromium-based browsers expose so that developer tools, browser automation frameworks, and testing suites can inspect and control a running browser instance. Historically, an attacker or red-team operator activated CDP the straightforward way: relaunch the browser with the `--remote-debugging-port` command-line switch, which opens a local TCP listener that speaks CDP over HTTP and WebSockets. Security researchers and malware authors have abused this flag to enumerate and steal cookies since at least

2018 [5], and dedicated proof-of-concept tools for doing so over the debugging port have circulated publicly for years [6]. Google's response, shipped in Chrome 136 in April 2025, made the default configuration harder to abuse: the browser now ignores `--remote-debugging-port` and `--remote-debugging-pipe` when they target the user's default profile directory, honoring them only when paired with a non-standard `--user-data-dir`, which forces the browser into a freshly created profile encrypted under a different application-bound key than the user's real, logged-in session [5][6]. That same release cycle introduced App-Bound Encryption, which ties the operating-system-level key that protects Chrome's cookie database to the specific application binary and user account, closing off the long-standing technique of copying the SQLite cookie store and decrypting it offline [5].

Those two mitigations pushed attackers toward a harder but more durable approach: rather than launching a new, debuggable browser instance, activate the debugging server inside the browser process that is already running and already authenticated. DeathFlamingo's December 2025 research documented that Edge could be coerced into starting its internal remote-debugging server through direct manipulation of a live process, and Cedric Van Bockhaven's subsequent write-up, "Modern Session Hijacking by Living off the DevTools Protocol," generalized and refined the approach across Chromium-based browsers [1][2]. SpecterOps' August 2026 release operationalized that research into a Beacon Object File — a compact, position-independent code module commonly used with the Cobalt Strike post-exploitation framework — called CDP-Enable-BOF, paired with a companion CDP Toolkit client for issuing commands once the debugging server is live [1][2]. This body of work sits alongside Google's own March 2025 observation that attackers were increasingly turning to Chrome remote debugging as disk-based cookie theft became more difficult, and it directly informs how defenders should read the security value of DBSC, the newer hardware-bound session protection Google has been rolling out through 2026 [3][5].

Security Analysis

CDP-Enable-BOF works by locating the target browser's process and the loaded Chromium module (`chrome.dll` or `msedge.dll`), then resolving the internal, unexported function responsible for starting the remote-debugging server — `StartRemoteDebuggingServer` — by matching byte-level signatures rather than relying on exported symbols, since Chromium does not expose this function through its public API [1][2]. The tool allocates a small stub of shellcode and an accompanying data block inside the target process's memory, installs a temporary Windows message-handling procedure on one of the browser's own windows, and uses that window's message queue to force execution of the stub on the browser's UI thread rather than through a conventional `CreateRemoteThread` call into an arbitrary thread [1]. Executing on the UI thread matters operationally because Chromium's internal state,

including the debugging-server toggle, is only safe to mutate from that thread. In this note's analytical assessment, driving execution through the browser's own message loop also plausibly reduces the likelihood of tripping Control Flow Guard, thread-local-storage validation, or Hardware-Enforced Stack Protection (CET) checks compared with a naive remote-thread injection, though neither SpecterOps nor The Hacker News state this specific evasion comparison directly, and it should be read as an inference from the mechanism described rather than a vendor-confirmed finding. Once the stub runs, the browser's existing, already-authenticated debugging server comes online — reachable locally or, if the operator proxies the port, from elsewhere on the network — and the toolkit can issue standard CDP calls such as `Storage.getCookies` to retrieve live session cookies directly from browser memory, along with browsing history, bookmarks, installed extensions, saved-password metadata through the autofill subsystem, and full interactive screen control via CDP's screencast functionality [1][2]. SpecterOps tested the chain against Chrome 147.0.7727.102 and Edge 147.0.3912.98, and noted that later Chromium releases in the 151.x line may require the tool's byte signatures to be updated, since the technique depends on matching unexported internal functions that Google is free to change between versions [1].

The DBSC interaction is the part of this disclosure most relevant to defenders who have already invested in Google's newer protections. DBSC generates a device-bound asymmetric key pair at login, stores the private half in hardware-backed storage such as a TPM when available, issues the browser only short-lived session cookies, and requires the browser to prove possession of that private key before Chrome will refresh an expired cookie — a deliberate design to make a stolen cookie useless once it has expired on any device other than the one it was issued to [3][4]. That design assumption holds precisely as intended against an attacker who exfiltrates a cookie and tries to replay it, or refresh it, from a second machine. It does not address an attacker who never leaves the original device: CDP-Enable-BOF does not extract, copy, or attempt to use the DBSC private key at all, and SpecterOps was careful to note in its research that its published work does not describe extracting that key [2]. Instead, the technique treats the entire authenticated browser as the credential. Any CDP call issued through the enabled debugging server executes inside the browser's own security context, so requests carry whatever cookies, including DBSC-protected ones, the browser is already presenting to the site — no replay, no key extraction, and no violation of DBSC's cryptographic guarantee is required, because the "device" DBSC is bound to is still the one making the request. This is consistent with the broader lesson from Google's own blog post introducing DBSC, which frames the protection as raising the cost of cookie theft and replay specifically, not as a general defense against an attacker who already has arbitrary code execution on the endpoint [3]. Independent commentary published after DBSC's general availability made the same point: device binding closes a common attack path (offline cookie replay) while leaving an authenticated, actively-used session on the compromised device itself just as valuable to an attacker as it always was [7].

Recommendations

Immediate Actions

Security teams should treat this disclosure as an update to endpoint detection coverage rather than a browser vulnerability requiring a vendor patch, since no CVE applies and the technique depends on an attacker already having code execution on the host. Organizations running Sysmon or equivalent endpoint telemetry should confirm they are collecting and alerting on Event ID 8 (CreateRemoteThread) and Event ID 10 (ProcessAccess) with attention to access masks consistent with process injection — SpecterOps specifically flagged access mask `0x143a`, which combines thread-creation, virtual-memory, and query-permission rights characteristic of this technique — where the target process is `chrome.exe` or `msedge.exe` [2]. Incident responders should also confirm that any host suspected of broader compromise is checked for unexpected local or proxied listeners on non-standard ports bound to browser processes, since CDP-Enable-BOF's activated debugging server is the artifact most directly observable on the wire.

Short-Term Mitigations

Beyond detection tuning, organizations should verify that Chrome and Edge are patched to at least the 136-series baseline that enforces the `--user-data-dir` requirement for remote debugging switches, since that protection remains a meaningful barrier against the simpler, launch-time variant of cookie theft that predates this technique [5][6]. Endpoint protection and EDR policies should be updated to flag window-message-based code execution targeting browser processes, not just classic `CreateRemoteThread` injection, since this is the specific mechanism CDP-Enable-BOF uses to evade more common injection signatures [1]. Security awareness and incident-response playbooks for privileged users — administrators, developers with elevated SaaS access, and finance or HR staff with access to sensitive web applications — should be updated to reflect that an attacker with code execution on their workstation can take over any actively open, authenticated browser session regardless of DBSC, cookie encryption, or MFA already completed for that session, because MFA authenticates the login event rather than the ongoing session, and CDP access occurs entirely after that authentication has already succeeded.

Strategic Considerations

This disclosure reinforces that browser-session protections such as App-Bound Encryption and DBSC are valuable, complementary layers rather than substitutes for preventing and detecting endpoint compromise in the first place. Organizations should continue rolling out DBSC where available, since it does close the offline-replay and cross-device-refresh attack path that has historically been a common vector for commodity cookie theft, while recognizing in their risk models that DBSC provides no protection once an attacker has arbitrary code execution on the endpoint itself [3][4]. Longer term, security architecture teams should weigh session-length and re-authentication policies for high-value SaaS and identity-provider sessions against the reality that a sufficiently privileged endpoint compromise can hijack any session currently open in the browser, and should factor endpoint detection and response coverage for browser-process injection into the same control set they already apply to credential-theft and lateral-movement tooling generally.

CSA Resource Alignment

This finding extends a pattern CSA's AI Safety Initiative has tracked repeatedly through 2026: a hardened, patched credential-protection mechanism gets bypassed not by breaking the cryptography but by operating through an already-authenticated context that the protection was never designed to cover. CSA's rapid-research note on the VS Code Zero-Day: One-Click GitHub Token Theft analyzed a closely analogous dynamic in the github.dev webview sandbox, where a broadly scoped OAuth token was exposed to exfiltration once an attacker could execute code inside the editor's trust boundary; both cases illustrate that browser-hosted and browser-embedded execution environments concentrate session value in ways that a single control point — a webview sandbox in that case, DBSC's device key in this one — cannot fully contain once code execution is achieved nearby. CSA's Using Zero Trust to Counter Identity Spoofing & Abuse is directly relevant to the recommendations above: its guidance on continual authentication, unique session identifiers paired with inactivity timeouts, and identity threat detection and response (ITDR) tooling describes exactly the detective and session-hygiene controls that remain necessary even after a device-binding protection like DBSC is deployed, because that guidance assumes — correctly, as this disclosure shows — that session abuse from an already-trusted endpoint requires its own monitoring layer. Finally, this incident aligns with the spirit of the Identity and Access Management domain of the CSA AI Controls Matrix (AICM) v1.1, whose session-management and endpoint-integrity controls call for organizations to treat session tokens as monitored assets with a defined lifecycle rather than as artifacts of a one-time authentication event; teams building AICM-

aligned control evidence for browser-based access to AI development tools and SaaS platforms should incorporate browser-process injection detection into that evidence base alongside existing credential-theft monitoring.

References

[1] The Hacker News. "Chrome DevTools Technique Enables Authenticated Session Hijacking in Live Windows Browsers." The Hacker News, August 14, 2026.

[2] SpecterOps. "Return of the Cookie Monster." SpecterOps Blog, August 13, 2026.

[3] Google Security Blog. "Protecting Cookies with Device Bound Session Credentials." Google, 2026.

[4] Google Workspace Updates. "Prevent Account Takeovers with Device Bound Session Credentials (DBSC), Now Generally Available in the Chrome Browser for Windows." Google Workspace Updates Blog, May 2026.

[5] Chrome for Developers. "Changes to Remote Debugging Switches to Improve Security." Chrome for Developers Blog, March 2025.

[6] Red Canary. "Stealers Evolve to Bypass Google Chrome's New App-Bound Encryption." Red Canary Threat Intelligence, 2026.

[7] Constella. "Google Just Fixed Session Cookie Theft in Chrome. Here Is What It Still Cannot Stop.." Constella, 2026.

