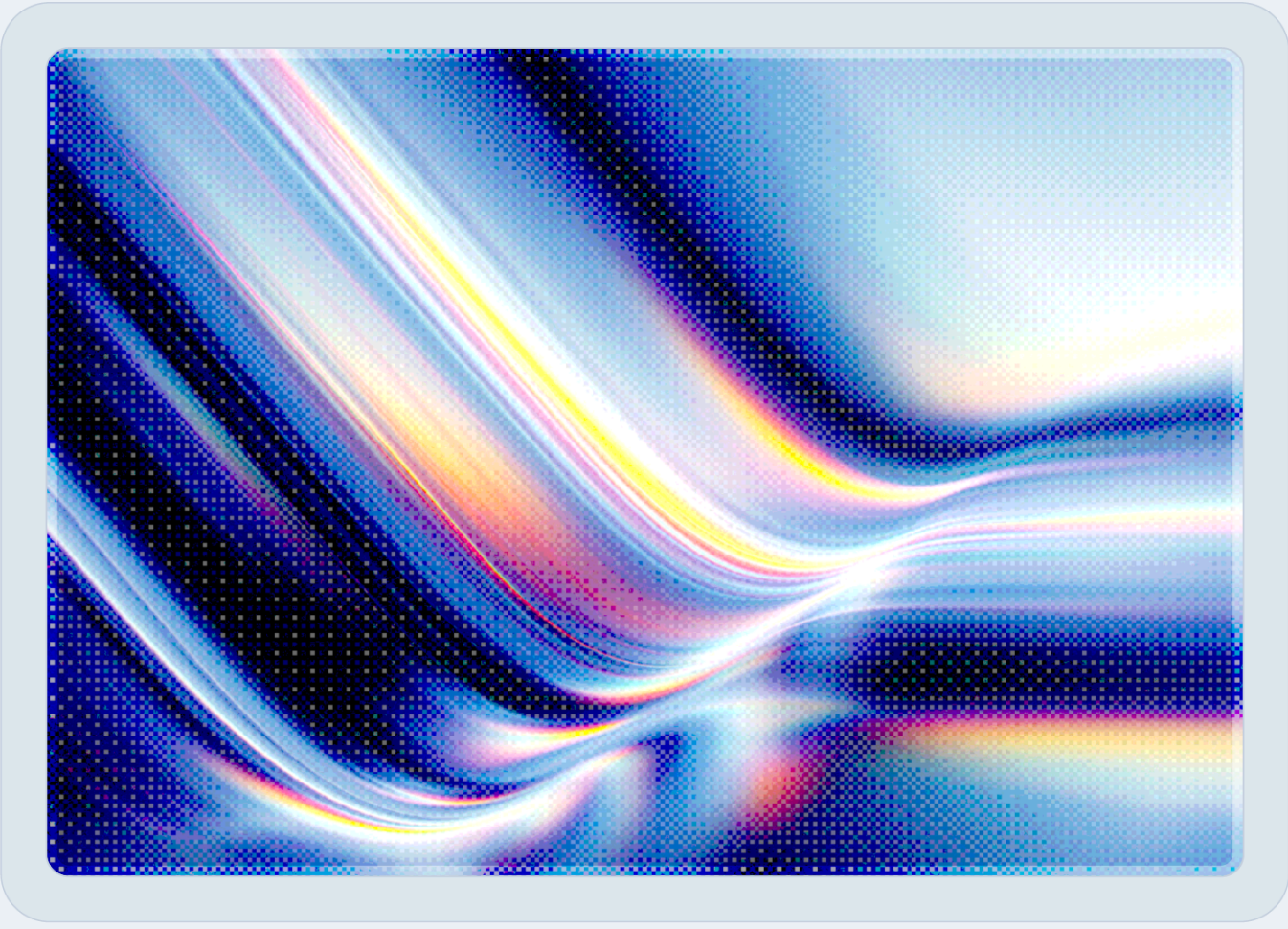
CSAI Foundation | Cloud Security Alliance

Someone Else's Registrar: Fourth-Party Code in Signing Contexts

Security Implications and Guidance After the Polymarket–DebugBear and StepDrainer Incidents

2026-11-01

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

In the last week of October 2026, two separate incidents exposed the same weakness. On October 29, someone replaced a real-user-monitoring (RUM) script that the performance vendor DebugBear served from its own CDN hostname. The new code used Polymarket users' authenticated trading sessions to sign and submit transfers and sell orders, and about $2.5 million was traced to the attacker's wallet [1][2]. Two days later, at least 12 cryptocurrency and DeFi front ends were redirected to wallet-drainer pages. Kintsu, one of the victims, said its domain had been compromised at the registrar level [6]. Root causes have been disclosed for only some of these cases. In those cases, neither attack relied on a flaw in the victim's own code, its smart contracts, or its users' keys.

- **The trust boundary of a signing page includes every party that can change what a script URL resolves to.** That covers the script vendor plus the vendor's registrar, DNS provider, CDN, and storage. These are fourth parties that the site operator has never assessed.
- **Host-allowlist Content Security Policies do not stop a trusted script from misusing trusted destinations.** The Polymarket drainer called Polymarket's own APIs, its RPC provider, and its swap service, and the policy allowed all of them [3]. Sources disagree on whether DebugBear's own hostnames were also allowed [2][3].
- **Observability, analytics, and marketing scripts do not belong on pages that hold a signer or session credential.** On those pages, any script that runs in the page's own context has the same authority as the user.
- **Registrar-level controls are now a front-end security control.** These controls include registry lock, registrar MFA, and DNS change monitoring. They apply to an organization's own domains and to the domains of the vendors whose code it loads.
- **DebugBear has not published a root cause for its compromise as of this writing.** Organizations should treat attribution claims as unconfirmed, including claims of DNS or registrar involvement [2][4][5].

Background

Each of the two October 2026 incidents compromised a different layer of the infrastructure that delivers code to users' browsers. Both reached the same end state: attacker-controlled code ran in a context where users could sign transactions. This section reconstructs each incident from the public record and then places both in the longer history of front-end supply chain attacks against cryptocurrency platforms.

The Polymarket–DebugBear Compromise

Polymarket embedded DebugBear's RUM snippet, a JavaScript file that measures page performance for real visitors. Each DebugBear customer loads its own project-specific file, and Polymarket's was `cdn.debugbear[.]com/C7xcKKrjUnXK.js` [3]. On October 29, 2026, Hudson Rock researchers retrieved a 15,187-byte version of that file. The script looked in local storage for Polymarket's authentication context, including the CLOB API key map and the proxy wallet address. It then searched React's internal component tree for an object that exposed trading methods such as `postOrder` and `createOrder` [2]. Next, it built EIP-712 typed-data batch messages on Polygon, asked the page's existing signer to sign them, and submitted them through Polymarket's own relayer [2]. What happened next depended on the account type. For externally owned accounts (EOAs), the script transferred position tokens out. For accounts created through an embedded wallet provider, it sold positions at progressively worse prices and then moved the proceeds [2].

The timeline of the attack and the response comes largely from independent reconstruction. Funds began flowing from victim accounts at about 17:10 UTC. Polymarket built a production release that removed the DebugBear script at 17:59:20 UTC, about 49 minutes later [3]. The exchange then went into cancel-only mode, and trading resumed within roughly an hour [1][3]. DebugBear posted that it was "impacted by an ongoing security incident" and told all RUM customers to remove the embed code [5]. Polymarket attributed the incident to a compromised third-party provider and committed to reimbursing affected traders [1]. BleepingComputer traced about 28.6 million POL, roughly $2.57 million, from the attacker wallet to three further wallets [1]. Other outlets have published estimates of about $2.7 million across 737 accounts [4][18]. One timeline reconstruction, by contrast, traced funds from 33 deposit wallets [3]. The final loss figure and the number of affected users are therefore not yet settled.

The initial access vector is still unknown. DebugBear has not said how its systems were breached or when the breach began [4]. Hudson Rock found no evidence that the malicious code reached other DebugBear customers [2]. One independent timeline reconstruction reports that the clean RUM file was last modified on October 7. The same source reports that the drainer response lacked the cloud storage headers the clean file carried, and that a different server answered for both DebugBear hostnames [3]. This evidence

is consistent with a change at the DNS, CDN, or origin-routing layer rather than an overwrite of the stored file. It does not establish either explanation. The source also records that DebugBear's domains later returned NXDOMAIN [3], which could reflect containment or further disruption.

The StepDrainer Registrar Wave

The second incident began on October 31, when the token launch platform Flap warned users that its domain had been hijacked. Blockaid linked the activity to the StepDrainer toolkit, detected similar activity at Kintsu, and then identified 10 additional sites serving it, including StonkFun, Pickle Finance, and Origami Finance [6]. The targets included both DeFi protocols and token launch platforms. Kintsu said its domain had been compromised at the registrar level. Flap's root cause was still under investigation when the coverage was published, and investigators had not established whether the affected domains shared a single point of compromise [6]. In each case, users who typed the correct domain received a drainer page that requested token approvals or signatures. No blockchain protocol was compromised [6].

A Recurring Pattern

These incidents fit a pattern of front-end compromises that reaches back several years. In May 2025, Curve Finance reported that attackers had gained access to its registrar's systems and changed the DNS delegation for its `.fi` domain. Curve stated that the hijack was unrelated to any breach of its own infrastructure and that there were no signs of unauthorized access or compromised login credentials [7]. In June 2024, Sansec reported that the new owner of the `polyfill.io` domain was injecting malware into more than 100,000 sites that loaded the library from its CDN [8]. In December 2023, attackers phished a former employee's npm account and used it to publish malicious versions of Ledger Connect Kit [10]. Most dApps loaded the kit at runtime from a CDN through Ledger's loader, which was designed to always fetch the most recent version. The malicious code therefore reached them dynamically, without any action on their part [19]. Ledger estimated user losses at about $600,000 [10]. The largest case came in February 2025. Investigators found that attackers had compromised a Safe{Wallet} developer machine and used leaked cloud credentials, an AWS S3 or CloudFront key, to change JavaScript served from Safe's storage. The modified code altered the transaction Bybit's signers approved and led to a theft of about $1.5 billion [9].

The table below summarizes these incidents together with the two October 2026 events, showing which layer of the delivery chain each attack compromised and what context the attacker then abused.

Incident	Date	Layer compromised	Context abused
Ledger Connect Kit	Dec 2023	Package registry account (npm)	dApp wallet connection and approvals [10][19]
polyfill.io	Jun 2024	Domain ownership transfer	General page context, redirects (not a signing context) [8]
Safe{Wallet} / Bybit	Feb 2025	Vendor cloud storage via developer machine and leaked credentials	Multisig transaction signing [9]
Curve Finance	May 2025	Registrar systems (DNS delegation)	Front-end approvals [7]
Polymarket / DebugBear	Oct 2026	Vendor script delivery (vector unconfirmed)	Existing authenticated session and signer [1][2]
StepDrainer wave	Oct 2026	Registrar/DNS for victim domains (confirmed for Kintsu)	Wallet approvals and signatures [6]

Taken together, these cases show that attackers no longer need to compromise a dApp's own code or its users' devices. Changing what a trusted URL serves is enough, and many parties can make that change.

Security Analysis

The incidents above raise three related questions. The first is whose infrastructure actually determines what code runs on a sensitive page. The second is why that matters more on pages that hold signing authority. The third is why the browser controls most organizations rely on did not contain the damage. The analysis closes with what these dynamics mean for AI agents that operate in browsers.

Why "Fourth-Party" Is the Right Frame

Third-party risk questionnaires typically focus on the vendor whose code a page loads. They seldom reach the infrastructure that decides what bytes arrive when a browser requests that vendor's URL. That chain includes the vendor's domain registrar, its authoritative DNS provider, its CDN and origin storage,

the cloud identities with write access to that storage, and its build and publishing pipeline. From the site operator's point of view, each link is a fourth party, and any one of them can replace the script. In the polyfill.io case, the replacement came through a lawful domain sale [8]. In the Safe{Wallet} case, it came through a cloud credential [9]. The browser cannot tell either kind of change apart from a legitimate update, and operators that do not pin or monitor scripts cannot either.

The title of this note refers to that gap. An organization can harden its own registrar account with registry lock and hardware-key MFA and still run another company's code on its most sensitive pages. That company's registrar, DNS, and storage controls then set the effective limit on the organization's security. The same class of compromise also hits operators directly. In the Curve and Kintsu cases, attackers went through the operator's own registrar [6][7]. The StepDrainer wave shows the first-party version of the risk, and the Polymarket incident shows the delegated version. In both, users were attacked at the correct URL.

Signing Contexts Weaken the Last Line of Defense

The consequences of script injection depend on what the compromised page can do. Injection on a content page typically enables credential phishing, payment-card skimming, or ad fraud. Injection on a page that holds a live session and a signer produces direct, final transactions. The Polymarket drainer did not need the user's password, seed phrase, or private key, and Hudson Rock found no routine that sent those materials to the attacker [2]. The script had no use for them. The page already held an authorized signer, and the relayer accepted the messages that signer produced. Inside the page's JavaScript realm, injected code has the same authority as the application's own code. It can call the same methods, read the same storage, and request the same signatures.

This shared authority limits what user-facing safeguards can achieve. Wallet confirmation prompts protect users only when a human reviews the request. Public analyses do not yet say whether Polymarket's EOA users saw wallet prompts. For embedded-wallet accounts, which sign without per-transaction confirmation, no human review stood between the injected script and the signer [2]. Even when a confirmation does appear, the Bybit case shows a related risk through a different mechanism. There, compromised front-end code made an altered multisig transaction look routine to experienced signers [9]. Session-based trading APIs exist to remove friction from signing, and they remove that friction for an attacker who shares the session as well.

The Limits of Allowlist-Based Controls

Content Security Policy (CSP) is the main browser control for limiting where scripts can load from and where they can send data [12]. Most deployments use a host allowlist, which protects against injection from attacker-controlled domains. It does not protect against a trusted domain serving untrusted content, or against trusted code misusing trusted destinations. One reconstruction states that every host the drainer called, apart from DebugBear's own, was allowed by Polymarket's CSP [3]. Those hosts included Polymarket's own APIs and relayer, its RPC provider, and its swap-quote service. A separate scan reported that the DebugBear host was also allowed [2]. The drainer sent its telemetry to DebugBear's `data.` subdomain [1]. If that host was not allowed under an enforced `connect-src` directive, the browser should have blocked the exfiltration. That this did not happen would suggest the policy was report-only, lacked `connect-src`, or treated script loading differently from connections. Public sources do not establish which. Either way, the trading actions themselves went only to destinations the policy allowed. A host allowlist could not have told the drainer's requests apart from the application's.

Subresource Integrity (SRI) takes a different approach. It binds a script tag to a cryptographic hash, and the browser refuses to execute bytes that do not match [11]. Had Polymarket pinned the file with an SRI hash, the browser would have refused the modified version. However, many SaaS snippets are designed to change without customer involvement, and vendors often update per-customer RUM and analytics loaders in place. Operators therefore face a choice. They can self-host and pin a reviewed version and accept the cost of maintaining it, or they can accept vendor-controlled updates and the fourth-party exposure that comes with them. The Ledger case shows the cost of the second option. Ledger's loader fetched the latest Connect Kit from a CDN at runtime, so the malicious versions reached dApps without any action on their part [19].

Regulation already applies this reasoning in one domain. PCI DSS v4.0 Requirement 6.4.3 calls for an inventory, authorization, and integrity assurance for every script on a payment page. Requirement 11.6.1 calls for detecting unauthorized changes to those scripts and to security-relevant headers [13]. In CSA's assessment, wallet signing pages, trading interfaces, administrative consoles, and OAuth consent screens carry comparable or greater risk. We are not aware of an equivalent mandate for these pages.

Implications for Agentic and AI-Integrated Front Ends

These dynamics extend to AI systems that act in browsers. Some agents operate through authenticated browser sessions. Some AI-powered applications hold delegated OAuth tokens or transaction-signing capabilities in the client. Both have the same exposure as the Polymarket front end. Any script loaded into that context can read the agent's session and act as the agent. Agents may also sign or approve requests without the human review that sometimes catches an anomaly. Organizations that deploy browser-based

agents should therefore treat every page an agent authenticates to as a signing context for the purposes of this analysis. This recommendation is an inference from the incident mechanics, not an observed attack pattern.

Recommendations

The recommendations below are grouped by time horizon. Immediate actions address direct exposure to the October 2026 incidents and the most obvious gaps in script hygiene. Short-term mitigations restructure how signing pages load and monitor code. Strategic considerations address vendor governance and signing architecture over the longer term.

Immediate Actions

Organizations that embedded DebugBear's RUM snippet should confirm that they have removed it, as the vendor advised [5]. They should check web and CDN logs for requests to `cdn.debugbear[.]com` and `data.debugbear[.]com`, and hunt for the published file hash and attacker wallet indicators [2]. Polymarket users who traded on October 29 should review their account activity against Polymarket's reimbursement guidance. Kintsu advised users who connected wallets or signed transactions on its site after 00:00 UTC on October 31 to revoke token approvals [6]. Users of the other domains Blockaid listed should take the same step and move exposed assets. Because hijack start times for those domains have not been published, they should review activity from October 30 onward to be conservative.

Beyond incident-specific response, security teams should identify every authenticated or signing page they operate. For each page, they should list every external script it loads, including scripts injected by tag managers. Teams should remove analytics, RUM, session replay, A/B testing, chat widgets, and advertising scripts from pages that hold signers, transaction APIs, or privileged sessions. The only exception is a documented business need that cannot be met another way. For their own domains, teams should confirm three things. Registry lock should be enabled where the TLD supports it, registrar accounts should require phishing-resistant MFA, and registrar and DNS change notifications should go to a monitored channel.

Short-Term Mitigations

Over the next quarter, organizations should treat signing pages as a separate security zone with a stricter script policy than the rest of the site. Any third-party code that must remain should be self-hosted at a pinned, reviewed version with SRI hashes [11]. A strict CSP should replace host allowlists with nonces or hashes and include `connect-src` restrictions that block unknown exfiltration destinations [12]. If a vendor cannot support pinning, its script should run in a sandboxed, cross-origin iframe that cannot reach the signer or session storage.

Detection deserves the same attention as prevention. Organizations should extend client-side change detection, of the kind PCI DSS 11.6.1 requires for payment pages [13], to signing and administrative pages. Monitoring should alert on any change to script hashes, response headers, or resolved infrastructure for third-party script hostnames. External monitoring of vendor hostnames, covering DNS resolution, TLS certificate issuance, and response fingerprints, might have revealed the shift in serving infrastructure that independent researchers later reconstructed for DebugBear [3]. On the server side, transaction relayers and order APIs should add anomaly controls. These include velocity limits, destination allowlists for transfers, and step-up confirmation for liquidations or full-balance transfers. These controls assume the client may be compromised, so they do not depend on the front end alone.

Strategic Considerations

Vendor governance needs to reach further down the delivery chain. Questionnaires for any vendor whose code runs in an organization's authenticated contexts should ask about four areas: registrar and DNS protections, storage write-access controls, publishing pipeline integrity, and whether customers can pin versions with integrity hashes. Contracts should require prompt notification of incidents that affect script delivery. Notification alone is not enough, however. In this incident, DebugBear posted its public warning at 18:00:58 UTC, about a minute and a half after Polymarket had built a release that removed the script at 17:59:20 UTC [3]. For high-value signing pages, organizations should not rely on vendor notification as their primary detection signal.

Where possible, organizations should also move signing authority out of the general-purpose page. Options include hardware or out-of-band confirmation for high-value actions, transaction simulation displayed in a context that page scripts cannot change, and short-lived, narrowly scoped session keys. For wallets and agent platforms, the goal is a signing path in which the content of each request is shown and approved outside the JavaScript realm that third-party code can reach. Industry bodies should consider whether the script-integrity obligations PCI DSS applies to payment pages should extend to transaction-signing and privileged-session pages.

CSA Resource Alignment

Several CSA frameworks give organizations a structured way to put these recommendations into practice. CSA's **AI Controls Matrix (AICM) v1.1** [14] provides the control vocabulary. Its Supply Chain Management, Transparency, and Accountability (STA) domain covers the third-party and downstream supplier assessment that, as this note argues, should extend to fourth-party infrastructure. Its Application and Interface Security (AIS) and Threat and Vulnerability Management (TVM) domains cover the integrity verification and change monitoring of client-delivered code recommended above. AICM also addresses AI-specific supply chains, which makes it relevant to the agentic front-end concerns raised in the Security Analysis.

The **SaaS Security Capability Framework (SSCF)** [15] defines security capabilities that SaaS vendors should expose to their customers. These include change control and configuration management, logging and monitoring, and security incident management. RUM and analytics providers are SaaS vendors whose product is code that runs in the customer's origin. The SSCF's approach to third-party risk management supports adding questions on version pinning, integrity hashes, and delivery-infrastructure change notification to SaaS vendor assessments.

CSA's Zero Trust guidance, particularly **Map the Transaction Flows for Zero Trust** [16], recommends mapping how sensitive transactions flow across components before setting protections. Applying that method to a browser signing flow makes the fourth-party dependencies visible. Every script origin, DNS zone, and CDN that the flow implicitly trusts becomes part of the protect surface. Finally, **Software Transparency: Securing the Digital Supply Chain** [17] covers inventory and provenance practices such as SBOMs. Those practices apply to scripts loaded at runtime just as they apply to packages compiled into a build. Client-side script inventories are meant to close that gap.

References

[1] BleepingComputer. "Polymarket halts trading after crypto-draining script breach, $2.5M traced." BleepingComputer, October 29, 2026.

[2] Hudson Rock / InfoStealers. "Polymarket Breach Alert: Wallet-Draining Code Found in a Third-Party Script." InfoStealers, October 29, 2026.

[3] Kirk / ohey.ca. "Polymarket wallet drainer served from DebugBear's script CDN." ohey.ca Research, October 29, 2026.

[4] Cyber Kendra. "Polymarket Hacked via DebugBear Script, Vendor Confirms Breach." Cyber Kendra, October 2026.

[5] DebugBear. "We're impacted by an ongoing security incident." X (formerly Twitter), October 29, 2026.

[6] Crypto Adventure. "StepDrainer Hits 12 Crypto Sites in DNS Hijack Wave." Crypto Adventure, October 31, 2026.

[7] Curve Finance. "Curve Domain Incident." Curve Finance News, May 19, 2025.

[8] Sansec. "Polyfill supply chain attack hits 100K+ sites." Sansec Research, June 25, 2024.

[9] BleepingComputer. "Lazarus hacked Bybit via breached Safe{Wallet} developer machine." BleepingComputer, February 26, 2025.

[10] The Hacker News. "Crypto Hardware Wallet Ledger's Supply Chain Breach Results in $600,000 Theft." The Hacker News, December 15, 2023.

[11] W3C. "Subresource Integrity." W3C Recommendation, June 2016.

[12] W3C. "Content Security Policy Level 3." W3C Working Draft.

[13] PCI Security Standards Council. "PCI DSS v4.x Document Library" (PCI DSS v4.0.1, Requirements 6.4.3 and 11.6.1). PCI SSC, 2024.

[14] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." CSA, 2025.

[15] Cloud Security Alliance. "SaaS Security Capability Framework (SSCF)." CSA, 2025.

[16] Cloud Security Alliance. "Map the Transaction Flows for Zero Trust." CSA Zero Trust Working Group, 2024.

[17] Cloud Security Alliance. "Software Transparency: Securing the Digital Supply Chain." CSA, 2022 (updated 2025).

[18] Crypto Adventure. "Polymarket Hack Drains $2.7M From 737 Wallets." Crypto Adventure, October 2026.

[19] Ledger. "Security Incident Report." Ledger Blog, December 20, 2023.