INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-26 18:38 UTC

Supply Chain Attack on Polymarket Frontend Vendor Drains $3M via Malicious JavaScript Injection

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0581
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Polymarket prediction market platform (polymarket.com); unnamed third-party frontend JavaScript vendor/dependency
Published	2026-06-26T14:04:12
Discovery Source	Rss

Executive Summary

Attackers compromised an unnamed third-party JavaScript vendor integrated into Polymarket's web application, injecting malicious code that manipulated blockchain transaction approvals and drained approximately $3 million from fewer than 15 users. The attack followed a classic supply chain pattern: the primary platform was not directly breached; instead, a trusted external dependency was poisoned to deliver malicious code through Polymarket's own interface. Any organization loading third-party scripts in a browser context with access to financial or authentication workflows faces the same exposure class.

Technical Analysis

The attacker compromised an unnamed third-party frontend JavaScript vendor integrated into polymarket.com. The injected script intercepted or manipulated wallet transaction approval flows within the browser, tricking users into signing fraudulent approvals that transferred control of their funds. Approximately $3 million in ParagonUSD (USDP) was drained, then converted to approximately 1,893 ETH and bridged from Polygon to Ethereum mainnet via cross-chain bridge, likely using proxy infrastructure (T1090.002) to obscure fund flow. No CVE has been assigned. Relevant CWEs: CWE-346 (Origin Validation Error), CWE-1357 (Reliance on Insufficiently Trustworthy Component), CWE-494 (Download of Code Without Integrity Check), CWE-79 (Cross-site Scripting). MITRE ATT&CK techniques include T1195.002 (Supply Chain Compromise: Compromise Software Supply Chain), T1059.007 (Command and Scripting Interpreter: JavaScript), T1204.001 (User Execution: Malicious Link), T1027 (Obfuscated Files or Information), and T1036 (Masquerading). No vendor

patch is available because the compromised component is unnamed; Polymarket has confirmed the incident and indicated full repayment to affected users. Patch status for the upstream vendor dependency is unknown.

Action Checklist

1. Step 1: Containment, Audit all third-party JavaScript dependencies currently loaded in your web application's browser context. Identify any scripts with access to wallet APIs, payment flows, or authentication token handling. Temporarily block or sandbox any unverified external scripts while the audit proceeds. Reference: NIST AC-20 (Use of External Systems), CIS 2.3 (Address Unauthorized Software).
2. Step 2: Detection, Review Content Security Policy (CSP) logs and browser-side error logs for unexpected script origins, inline script execution, or unauthorized external domain calls. In your SIEM, query for anomalous outbound connections from your web application's JavaScript delivery infrastructure. Behavioral indicator: unusual wallet approval transactions initiated from your platform's frontend without corresponding user-initiated actions. Reference: NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 8.2 (Collect Audit Logs).
3. Step 3: Eradication, Remove or replace any third-party JavaScript vendor confirmed to have been compromised. Implement Subresource Integrity (SRI) hashes for all externally loaded scripts so the browser rejects modified versions. Enforce a strict Content Security Policy that explicitly allowlists approved script sources and blocks inline execution. Reference: NIST CM-5 (Access Restrictions for Change), CWE-494 remediation guidance (Subresource Integrity verification).
4. Step 4: Recovery, Re-deploy your frontend from a verified clean build. Validate that all loaded scripts match expected SRI hashes. Monitor transaction approval flows post-deployment for anomalies. Confirm with affected users that wallet approvals have been revoked and re-issued where possible. Reference: NIST IR controls (Incident Response), AU-12 (Audit Record Generation) for ongoing transaction logging.
5. Step 5: Post-Incident, Conduct a full third-party dependency review against your software inventory. Establish a vendor security assessment process for all frontend dependencies with access to sensitive browser contexts. Implement continuous integrity monitoring for externally loaded scripts. Reference: NIST AC-20 (Use of External Systems), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), NIST CM-3 (Configuration Change Control).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to legal counsel, executive leadership, and potentially FinCEN/SEC if active wallet-draining transactions are still occurring on-chain, if more than 15 users are identified as affected (expanding breach scope beyond the initially reported threshold), or if any affected user's losses trigger state-level money transmission or securities reporting obligations given Polymarket's prediction market classification.

Recovery Notes	Post-containment, redeploy the Polymarket frontend exclusively from a clean build artifact whose dependency tree has been fully audited against the pre-compromise package-lock.json and whose every external script is covered by an SRI hash enforced via a strict CSP. Monitor Polymarket's on-chain smart contract for anomalous ERC-20 `Approval` events and `eth_sendTransaction` calls for a minimum of 30 days post-redeployment, as attacker-planted persistence mechanisms in browser localStorage or service worker caches may trigger delayed wallet interaction for users who have not cleared their browser state. Direct all affected users to revoke token approvals immediately using revoke.cash or a direct `approve(spenderAddress, 0)` call, and validate revocation on-chain before considering user-side recovery complete.
Forensic Artifacts	Malicious vendor JavaScript bundle: the exact file as served from the third-party CDN during the attack window, preserved with HTTP response headers (ETag, Last-Modified, Content-Length) — this file contains the wallet-approval interception logic that monkey-patched window.ethereum.request or eth_sendTransaction to redirect approvals to an attacker-controlled address. | On-chain Approval event logs: ERC-20 Transfer and Approval events emitted by the token contracts interacted with by Polymarket, filtered for the attacker's spender/drainer contract address, with block number, transaction hash, and wallet address of each victim — queryable via Polygonscan API or direct eth_getLogs RPC call. | CSP violation report JSON blobs: browser-generated reports captured by the CSP report-uri endpoint containing blocked-uri (the unauthorized script origin), script-sample (first 40 characters of the blocked inline script), and violated-directive fields that identify exactly when and how the malicious script attempted to execute outside the allowlisted policy. | Browser HAR (HTTP Archive) files: full network waterfall from production page load during the compromise window, showing the initiator chain from Polymarket's HTML to the third-party vendor bundle request, including any secondary fetches (XHR, WebSocket, fetch() calls) initiated by the malicious payload to exfiltrate wallet state or relay signed transactions. | npm registry and CDN publish history for the compromised vendor package: the exact version, publish timestamp, and diff between the last known-clean release and the malicious release — establishes the attack's introduction point in the supply chain and the precise exposure window for all downstream consumers of the package.

Per-Action IR Details

Step 1: Containment — Audit all third-party JavaScript dependencies currently loaded in your web application's browser context. Identify any scripts with access to wallet APIs, payment flows, or authentication token handling. Temporarily block or sandbox any unverified external scripts while the audit proceeds. Reference: NIST AC-20 (Use of External Systems), CIS 2.3 (Address Unauthorized Software).

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST AC-20 (Use of External Systems), NIST AC-4 (Information Flow Enforcement), CIS 2.3 (Address Unauthorized Software)

Compensating: Run `document.scripts` enumeration via browser DevTools console on live production page and export to JSON: `JSON.stringify([...document.scripts].map(s=>({src:s.src,async:s.async,integrity:s.integrity})))`. Diff output against last known-good deployment manifest. For teams without a CDN firewall, use a browser extension such as uMatrix or a local reverse proxy (mitmproxy) to immediately block fetches to the suspect vendor origin at the network layer while the audit proceeds.

Evidence: BEFORE blocking or sandboxing any external scripts, capture the full browser-loaded script inventory: export browser DevTools Network HAR file capturing all script resource fetches (including initiator chain), record CDN response headers (including ETag and Last-Modified for the compromised vendor bundle), and snapshot the live DOM via `document.documentElement.outerHTML` to preserve any injected inline script nodes. This volatile

browser-session state is destroyed the moment scripts are blocked or the page is redeployed. Also capture any active WebSocket or XHR connections visible in DevTools that the malicious script may have opened to exfiltrate wallet approval data.

Step 2: Detection — Review Content Security Policy (CSP) logs and browser-side error logs for unexpected script origins, inline script execution, or unauthorized external domain calls. In your SIEM, query for anomalous outbound connections from your web application's JavaScript delivery infrastructure. Behavioral indicator: unusual wallet approval transactions initiated from your platform's frontend without corresponding user-initiated actions. Reference: NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 8.2 (Collect Audit Logs).

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST AU-6 (Audit Record Review, Analysis, And Reporting), NIST AU-2 (Event Logging), CIS 8.2 (Collect Audit Logs)

Compensating: If no SIEM is available, query web server access logs directly using grep/awk: `grep -E '(eval|atob|fromCharCode|WebSocket|wallet|approve|signTransaction)' /var/log/nginx/access.log` to surface obfuscated JS execution patterns reaching the backend. For CSP violation reports without a reporting endpoint, configure `report-uri` to a free Sentry.io project or a local netcat listener (`nc -lvp 8888`) to capture browser-generated CSP violation JSON blobs in real time. Correlate wallet approval transaction hashes on-chain (Polygonscan or Etherscan) against Polymarket frontend session timestamps to identify transactions that lack a corresponding user-click event in application logs.

Evidence: Key artifacts specific to this supply chain JS injection: (1) CSP violation reports naming the unauthorized script origin domain — preserve raw report JSON including `blocked-uri`, `document-uri`, `script-sample`, and `violated-directive` fields; (2) Web application firewall (WAF) logs showing fetch requests to the third-party vendor's CDN endpoint, timestamped around the compromise window; (3) On-chain transaction records from Polymarket's smart contract showing `approve()` or `permit()` calls with anomalous spender addresses not matching Polymarket's official contract addresses; (4) Browser console error logs (exported from production error-tracking tools such as Sentry or Datadog RUM) containing stack traces referencing the malicious vendor bundle filename and line number.

Step 3: Eradication — Remove or replace any third-party JavaScript vendor confirmed to have been compromised. Implement Subresource Integrity (SRI) hashes for all externally loaded scripts so the browser rejects modified versions. Enforce a strict Content Security Policy that explicitly allowlists approved script sources and blocks inline execution. Reference: NIST CM controls (configuration management), CWE-494 remediation guidance, D3-FMBV (File Magic Byte Verification — adapted: integrity hash verification for scripts).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST AC-20 (Use of External Systems), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Generate SRI hashes for all retained external scripts using `openssl dgst -sha384 -binary vendor.js | openssl base64 -A` and embed as `integrity` attributes on each `` tag. To enforce CSP without a WAF, add `Content-Security-Policy` response headers directly in nginx (`add_header Content-Security-Policy "script-src 'self' https://trusted-cdn.example.com; object-src 'none'; base-uri 'self';"`) and validate enforcement using the free CSP Evaluator tool (csp-evaluator.withgoogle.com). Run YARA rules against the vendor's JavaScript bundle before re-integration to scan for known wallet-draining patterns (e.g., rules matching `eth_sendTransaction`, `wallet_switchEthereumChain`, or base64-encoded exfil URLs).

Evidence: BEFORE removing the compromised vendor script from production, preserve: (1) A bit-for-bit copy of the malicious vendor JavaScript bundle as served from the CDN (downloaded via `curl -o malicious_vendor_bundle.js https://vendor-cdn.example.com/bundle.js` with response headers saved via `-D headers.txt`); (2) A deobfuscated version of the injected payload — run through js-beautify and capture the wallet-approval interception logic (specifically

any monkey-patching of `window.ethereum.request` or `eth_sendTransaction` handlers); (3) The vendor's signed release history (npm audit log or GitHub release tags) to establish the exact commit or package version at which the malicious code was introduced. These artifacts are required for threat intelligence sharing and potential law enforcement referral.

Step 4: Recovery — Re-deploy your frontend from a verified clean build. Validate that all loaded scripts match expected SRI hashes. Monitor transaction approval flows post-deployment for anomalies. Confirm with affected users that wallet approvals have been revoked and re-issued where possible. Reference: NIST IR controls (Incident Response), AU-12 (Audit Record Generation) for ongoing transaction logging.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-12 (Audit Record Generation), NIST AU-3 (Content Of Audit Records), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: For teams without a formal CI/CD pipeline, perform the clean redeploy from a local build machine that has never loaded the compromised vendor: run `npm ci` (not `npm install`) from a locked `package-lock.json` pinned to pre-compromise dependency versions, then verify build artifact hashes with `sha256sum dist/*.js` against values recorded before the incident. Post-deployment, monitor Polymarket's on-chain contract for any `approve()` calls where the spender is not Polymarket's official proxy address — use a free blockchain monitoring service (Tenderly, OpenZeppelin Defender free tier, or a custom Etherscan API alert) to alert in real time on anomalous approvals from the frontend's connected wallet addresses.

Evidence: Before instructing affected users to revoke wallet approvals (which alters blockchain state), document: (1) Current on-chain token approval state for each affected wallet address via `eth_call` to the ERC-20 `allowance()` function, recording spender address, approved amount, and block number; (2) Transaction hashes of all malicious `approve()` transactions initiated during the attack window, pulled from the affected contract's event logs (`Approval` events filtered by compromised spender address); (3) Screenshots or API exports of each user's wallet approval history from a block explorer (Polygonscan) timestamped before revocation. These records establish the blast radius and support both user notification and any regulatory disclosure obligations.

Step 5: Post-Incident — Conduct a full third-party dependency review against your software inventory. Establish a vendor security assessment process for all frontend dependencies with access to sensitive browser contexts. Implement continuous integrity monitoring for externally loaded scripts. Reference: NIST AC-20 (Use of External Systems), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), D3-SFA (System File Analysis — applied to script delivery integrity).

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST AC-20 (Use of External Systems), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Use `npm audit` and Snyk CLI (free tier) to generate a full dependency graph of all frontend packages, flagging any transitive dependency with access to browser `window` globals. For continuous script integrity monitoring without a commercial tool, configure a cron job that runs every 15 minutes to fetch each externally loaded script URL, compute its SHA-384 hash, and compare against a baseline manifest — alerting via email or Slack webhook on mismatch: `curl -s https://vendor.example.com/bundle.js | openssl dgst -sha384 -binary | openssl base64 -A`. Store the vendor security assessment checklist in version control alongside the `package.json` so dependency additions require documented sign-off.

Evidence: For the lessons-learned record and threat intelligence output, preserve: (1) The complete npm dependency tree (`npm ls --all > dep-tree-post-incident.txt`) showing the full supply chain path from Polymarket's direct dependencies to the compromised vendor package; (2) Timeline reconstruction correlating the vendor's compromised package publish timestamp (from npm registry history or the vendor's GitHub commit log) against the first malicious on-chain transaction, to determine the exposure window; (3) IOC package: malicious script hash (SHA-256 and

SHA-384), CDN delivery URL, injected function signatures (e.g., the monkey-patched `window.ethereum.request` interceptor), and attacker-controlled exfiltration or wallet-draining contract address — formatted for sharing via MISP or a structured threat intelligence platform.

Detection Guidance

Query your web application logs and CDN access logs for unexpected changes in JavaScript file hashes or unexpected new script source domains. In browser-side telemetry or Real User Monitoring (RUM) tools, look for script execution from domains not in your approved CSP allowlist. In blockchain transaction monitoring (if applicable), flag wallet approval transactions that were not explicitly initiated by confirmed user interaction flows. Behavioral indicators: wallet approval events at unusual hours, batch approvals from a small number of accounts, or approvals for token amounts inconsistent with normal platform activity. MITRE T1059.007 (JavaScript execution) and T1195.002 (supply chain compromise) are the primary technique signatures. No specific IOC hashes or IPs have been confirmed in available source reporting; indicators should be derived from your own dependency integrity baselines. Reference: NIST AU-6, AU-12, CIS 8.2.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`Not confirmed in available source reporting`	No specific malicious script URLs, domains, or file hashes have been publicly disclosed by Polymarket or identified in available T3 source reporting	LOW

Framework Mappings

MITRE-ATTACK

- **T1027** — Obfuscated Files or Information
- **T1566.002** — Spearphishing Link
- **T1195.002** — Compromise Software Supply Chain
- **T1090.002** — External Proxy
- **T1036** — Masquerading
- **T1059.007** — JavaScript
- **T1204.001** — Malicious Link
- **T1566** — Phishing

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AT-2** — Literacy Training and Awareness
- **SC-7** — Boundary Protection

- **SI-8** — Spam Protection
- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **CA-7** — Continuous Monitoring
- **CM-3** — Configuration Change Control
- **SI-10** — Information Input Validation
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A03:2021** — Injection

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **16.10** — Apply Secure Design Principles in Application Architectures
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1027	Obfuscated Files or Information	Defense-Evasion
T1566.002	Spearphishing Link	Initial-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1090.002	External Proxy	Command-And-Control
T1036	Masquerading	Defense-Evasion
T1059.007	JavaScript	Execution

Technique ID	Technique Name	Tactic
T1204.001	Malicious Link	Execution
T1566	Phishing	Initial-Access

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/polymarket-customers...	T3
Prediction market giant Polymarket hit by cyberattack, with company ...	https://www.techradar.com/pro/security/prediction-market-giant-poly...	T3
Polymarket confirms hackers stole $3M from users after third-party ...	https://thenextweb.com/news/polymarket-hack-3-million-stolen-third-...	T3
Polymarket Loses $3M In Frontend Hack, Then Promises Full ...	https://yellow.com/news/polymarket-frontend-hack-repayment	T3
Polymarket Security Issues: Third-Party Breaches and User ...	http://www.gopher.security/news/polymarket-security-issues-third-pa...	T3

Generated 2026-06-26 18:38 UTC by TJS Security Command Center