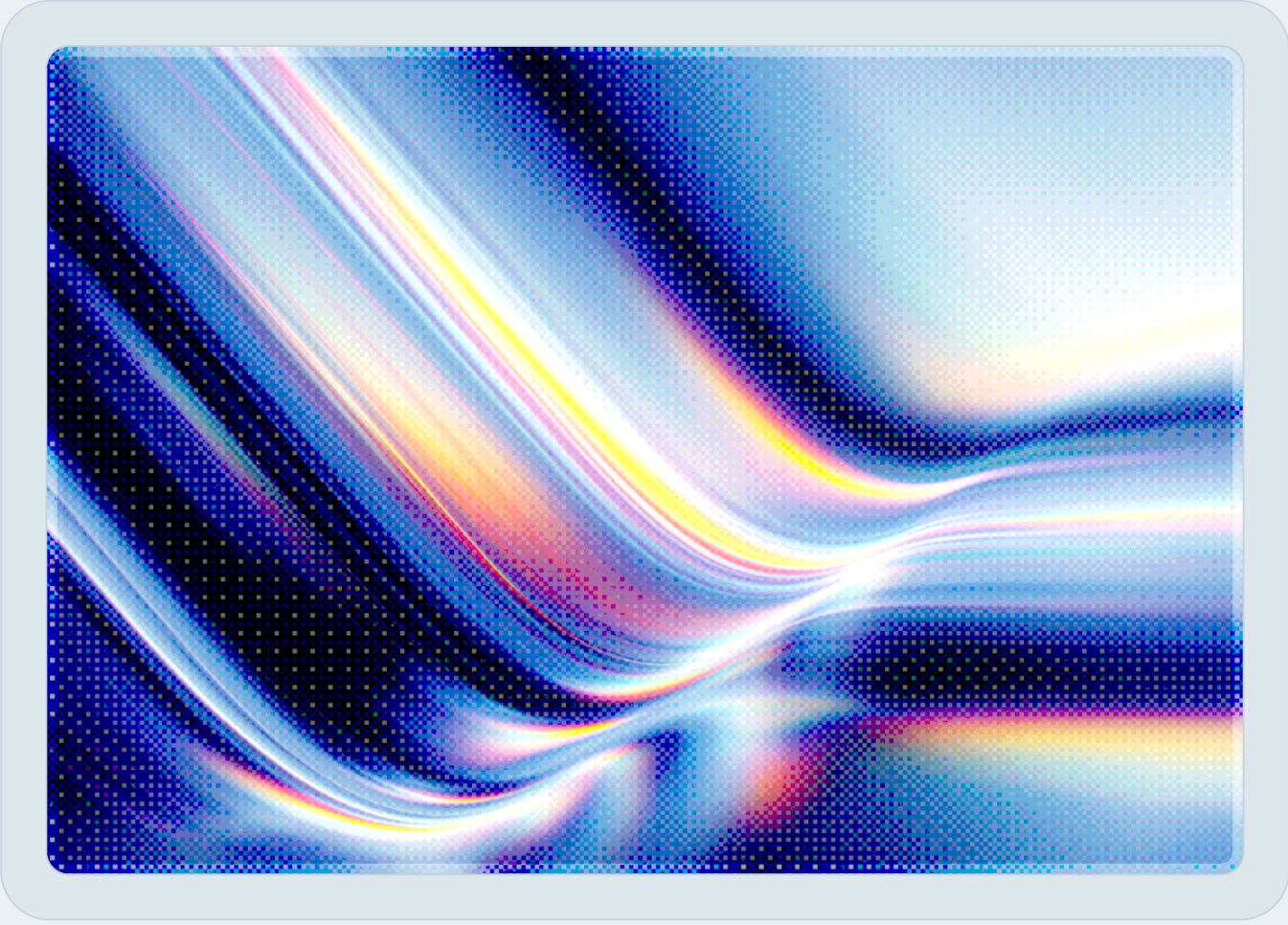
CSAI Foundation | Cloud Security Alliance

npm's indexed-btree Campaign Moves Malware to Runtime

2026-09-23

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

A malicious npm package named indexed-btree, along with nine to eleven related packages, was tied to more than 5.3 million cumulative downloads across the related-package family over roughly eleven weeks, a figure separate from indexed-btree's own registry-reported weekly download rate, before Checkmarx identified and reported the campaign in September 2026 [1][2]. Unlike the preinstall- and postinstall-script attacks that were common in npm supply chain incidents earlier in 2026, indexed-btree embedded its loader inside `BTree.prototype.set()`, a core method that legitimate applications call routinely during normal use, so the payload activated only when the library actually ran [1][2]. This design let the campaign operate without tripping install-time script restrictions of the kind npm introduced with its v12 lifecycle-script defaults, since the malicious behavior never executed during `npm install` at all [3]. Once triggered, the malware fingerprinted the host, beaconed to hardcoded Slack and Telegram channels, and retrieved an encrypted second-stage payload from a smart contract on the Ethereum Sepolia testnet using the EtherHiding technique, a blockchain-based command-and-control method previously associated with financially motivated and nation-state actors [2][4][9]. The operators reportedly earned approximately €231,000 (109 ETH) from the campaign, and the incident illustrates that install-time controls, however necessary, leave a runtime attack surface that organizations must address separately [1][2].

Background

Checkmarx published its analysis of a malicious npm package called indexed-btree, which impersonated the popular and legitimate sorted-btree library, on September 17, 2026; The Hacker News followed with its own report on September 22 [1][2]. The package was first published to the npm registry on June 18, 2026, by an account using the handle "charlessadler25," and npm's registry reported nearly two million weekly downloads for indexed-btree around the time it was pulled from the registry on approximately September 3, 2026 [2][5]. That weekly-downloads figure is a registry-reported snapshot rate, not a cumulative total, and it should not be added to the cumulative download counts described below. Checkmarx subsequently identified nine to eleven related malicious packages tied to the same operation, including btree-core, which alone drew an estimated 1.95 million cumulative downloads, and btree-time-index, with more than 425,000; combined, that family of related packages, excluding

indexed-btree's own separate weekly-rate figure, exceeded 5.3 million cumulative downloads, a reach comparable to other large npm compromises such as Shai-Hulud and Miasma documented earlier in the year [2].

The timing places indexed-btree squarely within the aftermath of one of npm's most significant recent trust-model changes. In June 2026, GitHub, which maintains the npm registry, announced that npm v12 would flip lifecycle scripts, Git-sourced dependencies, and remote-URL tarballs from a permissive default to deny-by-default, requiring maintainers to explicitly allowlist any script execution during installation [6]. That change was a response to a wave of 2025 and 2026 campaigns, including the Shai-Hulud worm family and the Miasma compromise of Red Hat's @redhat-cloud-services npm packages, that had relied on preinstall and postinstall hooks to execute code the moment a package was installed [6][7]. CSA's whitepaper on the campaign assessed both what npm v12's new defaults were designed to stop and where their coverage was likely to fall short [6]. The indexed-btree campaign, first published on June 18, 2026, the same week npm v12's defaults were being finalized for rollout, bears that assessment out: the package's design had the effect of bypassing install-script controls entirely, moving execution into a function ordinary application code calls constantly rather than into any hook npm v12 was built to restrict.

The identity behind the campaign has not been publicly attributed as of this writing. The npm account that published the packages maintained a GitHub presence, including a commit history and a profile image, that researchers described as credible-looking rather than bearing the hallmarks of a hastily assembled throwaway account; CSA assesses this presence was likely constructed to lend the account an air of legitimacy [1]. This social-engineering layer, combined with the package's functional mimicry of sorted-btree, allowed it to accumulate download volume comparable to an established open-source project before detection.

Security Analysis

A shift from install-time to runtime execution

The indexed-btree campaign's loader lived in an unusual place. Checkmarx researchers found that "this package does not rely on preinstall / postinstall at all. Instead, it runs entirely from application code at runtime" [2]. The malicious trigger sat inside `BTree.prototype.set()`, the method a developer calls every time they insert a value into the data structure, meaning the malware activated during ordinary, expected use of the library rather than during a one-time installation event [1][2]. This is a departure from the attack pattern that was common through 2025 and early 2026, in which threat actors like the Miasma operators weaponized preinstall hooks to execute a payload the instant `npm`

`install` ran, before a developer had any opportunity to review or use the package [6]. Because npm v12's deny-by-default posture specifically targets lifecycle scripts, Git-sourced dependencies, and remote-URL tarballs, a package whose malicious code never touches those mechanisms sails through the new controls untouched [3][6]. Checkmarx's own assessment of the technique underscored that "runtime behavior analysis, not just install-time, is what actually catches packages like this" [2], a conclusion that reframes install-script hardening as necessary but insufficient rather than as a comprehensive fix.

Payload architecture and blockchain-based command and control

Once triggered, the malware executed a separate obfuscated JavaScript file referred to in reporting as sharedLoad.min.js, which fingerprinted the host by collecting operating system details, hostname, CPU information, memory, and system uptime [1][2]. That data was exfiltrated through hardcoded Slack and Telegram channels, giving the operators a reporting channel built on infrastructure that ordinary network monitoring is less likely to flag as malicious [1][2]. For its second stage, the campaign used the EtherHiding technique, storing encrypted payload components inside a smart contract deployed on the Ethereum Sepolia testnet and retrieving them via the blockchain rather than a conventional domain or IP-based command-and-control server [2][4]. EtherHiding was first documented by Google's Mandiant and Google Threat Intelligence Group researchers tracking the financially motivated cluster UNC5142 [4], and Google's Threat Intelligence Group has since reported its use by the North Korea-linked group UNC5342 [9]. That progression, from a technique tied to one financially motivated cluster to adoption by a separate, state-linked actor, suggests EtherHiding is spreading beyond its original cybercrime origin; indexed-btree's own use of the technique is consistent with that spread extending further, to operators without a confirmed nation-state or established-cluster affiliation. The approach is attractive to attackers precisely because a public blockchain cannot be seized or taken down the way a hosting provider or registrar can act against a malicious domain; Checkmarx noted the same durability advantage in its analysis of the indexed-btree infrastructure [2]. The malware additionally used ECDH key exchange with a hardcoded X25519 public key to encrypt communications with its second-stage infrastructure, and after execution it deleted its own artifacts to reduce forensic traces [1][2].

Financial outcome and campaign scale

Blockchain analysis embedded in the reporting indicates the operators collected approximately 109 ETH, valued at roughly €231,000 at the time of reporting in September 2026, over the life of the campaign [1][2]. While the exact monetization mechanism, whether cryptomining, credential theft resale, or a downstream ransomware or wiper deployment, was not fully detailed in the initial disclosures, the combination of indexed-btree's registry-reported download footprint and a multi-month operating

window before detection illustrates how a runtime-triggered payload can plausibly sustain a longer dwell time than an install-time script, which security tooling focused on `postinstall` hooks or static `package.json` scans is generally better positioned to catch at install time.

Why this matters for AI and ML developer environments

CSA's whitepaper on the 2026 npm campaign wave identified the AI and ML developer toolchain as a disproportionate concentration point for npm supply chain risk, noting that AI-adjacent infrastructure such as LLM gateways tends to aggregate an unusually high density of sensitive credentials, including cloud provider keys, model-provider API tokens, and vector database administrative credentials, on build runners and developer workstations [6]. A data-structure utility like a B-tree implementation is a plausible transitive dependency in data pipeline, caching, or indexing code that AI and ML tooling commonly relies on, which means organizations running AI workloads should not assume that only obviously AI-branded packages carry AI-relevant supply chain risk. Any dependency capable of reaching a build runner or developer machine that also holds model-provider credentials is a potential vector, regardless of how mundane its stated purpose appears. This concentration risk is not hypothetical: CSA's research note on the DPRK-linked compromise of the Mastra AI npm ecosystem documented a separate attack, using compromised-maintainer dependency injection rather than a runtime-triggered loader, that targeted AI and ML developer environments for the same reason indexed-btree's fingerprinting and beaconing behavior would make it valuable there [10].

Recommendations

Immediate Actions

Organizations that installed indexed-btree, btree-core, btree-time-index, or any of the other packages identified in the Checkmarx disclosure between June 18 and September 3, 2026, should treat any environment that ran the affected code as compromised and rotate credentials accessible from those systems, including cloud provider keys, CI/CD tokens, and any model-provider or AI service API keys present on the same host [2]. Security teams should search dependency manifests and lockfiles for the specific package names identified in the Checkmarx report and cross-reference the published indicators of compromise, including the Sepolia smart contract address, RPC endpoints, and the Telegram bot token disclosed in the vendor writeup, against network and DNS logs [2]. Because the malware deletes its own artifacts after execution, absence of the dropped files on disk today does not indicate the host was never compromised; log-based and network-based indicators should take priority over filesystem inspection.

Short-Term Mitigations

Runtime behavior monitoring should be added as a complement to, not a replacement for, install-time script controls. Software composition analysis tools that only scan for suspicious `preinstall` or `postinstall` hooks will not catch a payload embedded inside a commonly called method like `BTree.prototype.set()`; organizations should evaluate whether their existing tooling performs any runtime or behavioral analysis of dependency code, and if not, treat that as a documented gap rather than an assumed capability. Outbound network egress from build runners and production environments should be restricted to known-necessary destinations; a default-deny egress posture would have blocked or at least flagged the beaconing to Slack, Telegram, and the Sepolia RPC endpoints this campaign relied on. Teams should also complete their npm v12 allowlisting work if they have not already, since the deny-by-default posture for lifecycle scripts, Git dependencies, and remote-URL tarballs still closes off the most common and historically most damaging install-time vector even though it does not address runtime-triggered payloads like this one [6].

Strategic Considerations

Detecting malware that activates during ordinary application use, rather than at a discrete installation event, requires a different class of tooling than install-time-focused software composition analysis, which CSA assesses remains the more common deployment pattern among organizations today. Static analysis of a package's declared scripts and manifest metadata cannot surface a trigger hidden inside a widely called method; catching this class of attack requires either dynamic analysis that exercises a package's actual code paths before it reaches production, or behavioral monitoring capable of flagging anomalous network activity, host fingerprinting, or blockchain RPC calls originating from application code that has no legitimate reason to perform them. Organizations building or expanding software supply chain security programs should treat runtime dependency behavior as a distinct control objective from install-time script governance, budgeting for both rather than assuming that npm v12-style defaults resolve the broader problem. Given that EtherHiding has now moved from a technique associated with sophisticated, well-resourced threat clusters to one apparently available to whoever operated the indexed-btree campaign, defenders should also expect blockchain-based command-and-control to appear more frequently in commodity supply chain malware going forward, and should ensure network detection capabilities are not blind to RPC traffic toward public blockchain infrastructure.

CSA Resource Alignment

This incident extends themes CSA has already documented in its 2026 npm supply chain research. The whitepaper "npm Supply Chain Under Siege: TeamPCP, Miasma, and npm v12" analyzed the campaign wave that led directly to npm v12's deny-by-default lifecycle-script controls and assessed, in detail, what those controls can and cannot protect against [6]. That paper's central argument, that provenance and install-time attestation mechanisms defend against a specific class of attack but leave other execution paths open, is precisely what the indexed-btree campaign demonstrates in practice: a package that never touches a lifecycle hook sidesteps controls built to police lifecycle hooks. Organizations that used that whitepaper to plan their npm v12 migration should treat indexed-btree as confirmation that the migration addresses one attack class among several, not full closure of the install-time and runtime supply chain risk surface.

CSA's research note "Miasma: Red Hat npm Supply Chain Worm" offers a useful point of contrast [7]. Miasma weaponized a preinstall hook to harvest multi-cloud credentials the instant a compromised package was installed, representing the install-time attack pattern that npm v12 was explicitly designed to close off. Reading the two incidents together shows the practical effect of npm's policy change: it raises the cost of the Miasma-style attack while leaving open the indexed-btree-style attack, reinforcing CSA's recommendation that supply chain security programs pursue defense-in-depth across identity, install-time, and runtime layers rather than relying on any single control.

CSA's research note "Sapphire Sleet Poisons Mastra AI npm Supply Chain" adds a third data point to this pattern, documenting a DPRK-linked compromise of the Mastra AI npm ecosystem that relied on compromised-maintainer dependency injection rather than either a malicious lifecycle hook or a runtime-triggered loader [10]. Read together, the three incidents show that AI and ML developer toolchains face npm supply chain risk through multiple, mechanically distinct attack paths, none of which a single control, whether install-time script governance or dependency provenance verification, can fully close on its own.

All three artifacts also connect to CSA's AI Controls Matrix (AICM) v1.1, available at cloudsecurityalliance.org/artifacts/ai-controls-matrix-v1-1, which organizations can use to evaluate dependency governance and runtime monitoring controls for AI and ML pipelines specifically, given that those environments concentrate the kind of credentials this campaign's fingerprinting and beaconing behavior was built to locate and exfiltrate [8].

References

[1] The Hacker News. "Malicious npm Package indexed-btree Hid Its Loader in Runtime Code Before Removal." The Hacker News, September 2026.

[2] Checkmarx. "npm 'btree' Malware Campaign Affects Millions of Downloads, No Need for Install Script." Checkmarx Zero, September 17, 2026.

[3] InfoWorld. "New npm malware finds a way around install script defenses." InfoWorld, September 2026.

[4] Google Cloud. "New Group on the Block: UNC5142 Leverages EtherHiding to Distribute Malware." Google Cloud Blog, October 2025.

[5] SecurityWeek. "Malicious B-tree NPM Package Accumulates Millions of Downloads." SecurityWeek, September 2026.

[6] Cloud Security Alliance. "npm Supply Chain Under Siege: TeamPCP, Miasma, and npm v12." CSA AI Safety Initiative, June 11, 2026.

[7] Cloud Security Alliance. "Miasma: Red Hat npm Supply Chain Worm." CSA AI Safety Initiative, June 3, 2026.

[8] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, June 22, 2026.

[9] Google Threat Intelligence Group / Mandiant. "DPRK Adopts EtherHiding: Nation-State Malware Hiding on Blockchains." Google Cloud Blog, October 16, 2025.

[10] Cloud Security Alliance. "Sapphire Sleet Poisons Mastra AI npm Supply Chain." CSA AI Safety Initiative, June 22, 2026.