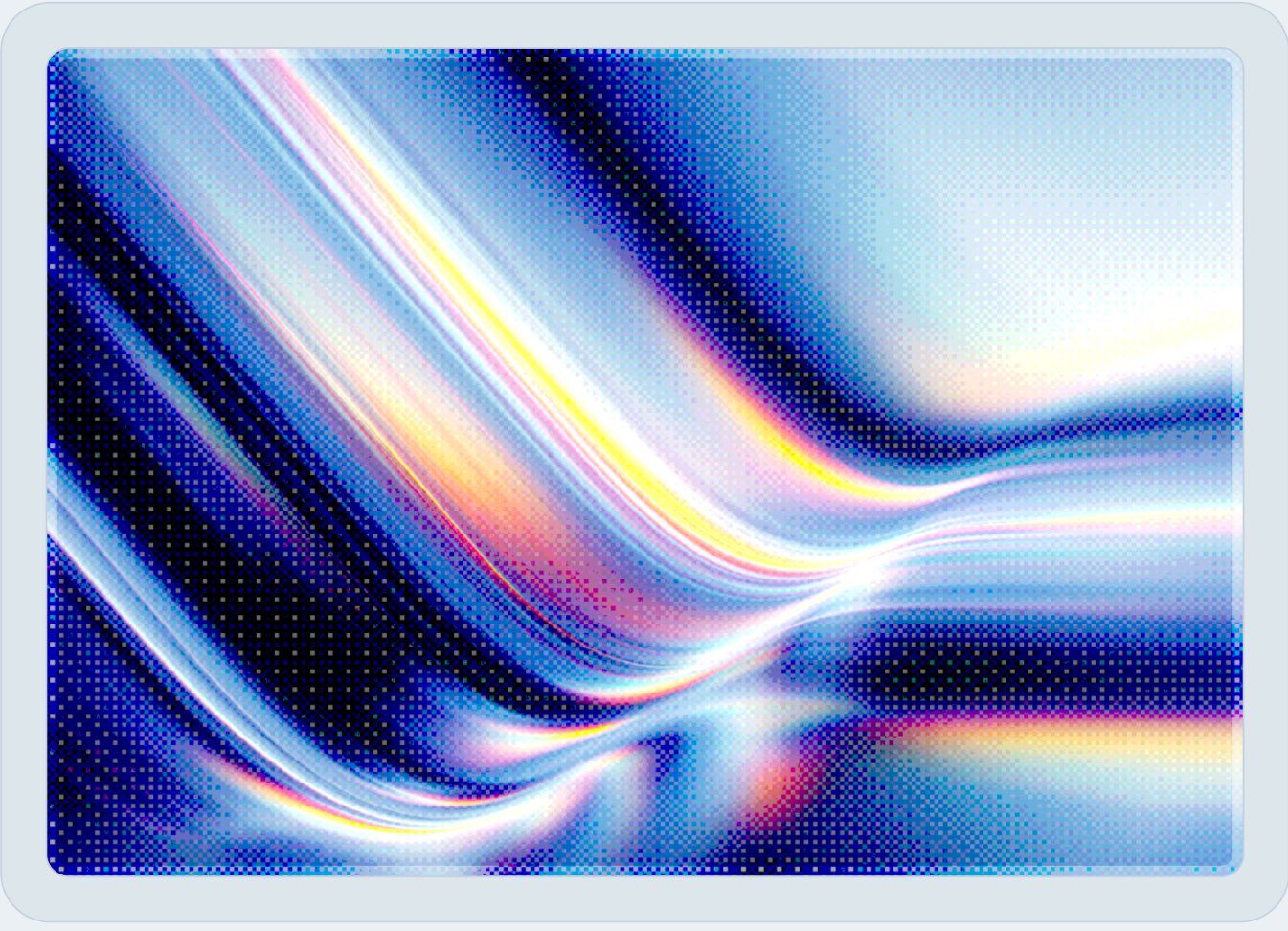
CSAI Foundation | Cloud Security Alliance

Flooding Dropper: Slop-Squatted npm Packages Deliver RAT

Nearly 800 Malicious Packages Target Windows, macOS, and Linux Developers

2026-08-11

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Security researchers have identified a large-scale malicious package campaign, tracked by Sonatype as "Flooding Dropper," that published nearly 800 packages to the npm registry between roughly mid-2026 and early August [1][3]. The packages deliver a cross-platform downloader called WEL1DROPPER that fetches operating-system-specific second-stage payloads for Windows, macOS, and Linux, culminating in remote access trojan (RAT) and infostealer capability on infected developer machines [1][2]. A researcher quoted in early reporting characterized the package names as appearing either "AI slop squatted, or randomly generated typo-squatting" — a description this note treats as an open question rather than a confirmed causal finding, since no source has yet demonstrated that the specific names were sourced from LLM-hallucinated suggestions rather than from an attacker-run generator [1]. Regardless of the precise name-generation method, the campaign is notable for abandoning the well-monitored `preinstall` / `postinstall` lifecycle hooks that registry scanners watch closely, instead instructing developers via package README files to manually load the code with `require()` [1][3]. The operation also builds on a documented lineage of prior npm supply chain campaigns targeting Russian financial infrastructure, suggesting an actor iterating on technique across multiple 2026 waves rather than a one-off event [4][1].

Background

npm has seen a series of malicious-package campaigns throughout 2026, of which Flooding Dropper is the latest, and by publicly reported package count the largest, rather than an isolated incident. In April and May 2026, a campaign later nicknamed "Moika" published 183 packages across at least three waves, each using npm's `postinstall` hook to exfiltrate the full `process.env` of the installing machine, download a second-stage reverse-SSH RAT from `oob.moika.tech`, and impersonate Russian financial services including Sberbank, Alfa-Bank, BCS, and the EMCD cryptocurrency exchange [4]. Researchers confirmed single-actor attribution across the Moika waves by identifying a shared hardcoded secret embedded in every stage of the payload, and noted that later waves added obfuscation, kill switches, and more structured command-and-control handshakes compared to the cleartext first wave [4]. That trajectory — added obfuscation, kill switches, and more structured command-and-control, and a deliberate move away from techniques defenders had learned to detect — sets up the Flooding Dropper campaign that followed.

Sonatype's research team began tracking Flooding Dropper (internally as sonatype-2026-005660) after researcher Paul McCarty at OpenSourceMalware flagged a package named `bigops-backend` on August 5, 2026 [3][1]. By the time Sonatype and The Hacker News published their analyses days later, the count of implicated packages had reached approximately 846 at the time of Sonatype's initial disclosure, later reported as roughly 850 as the count continued to grow, and characterized in headlines as "nearly 800" [1][3][5]. Rather than a small number of prolific publisher accounts, the campaign relies on a large number of freshly created npm accounts, each publishing only a handful of packages, with package names formed by combining recurring terms such as "bigops" and "bnpl" with other words and version numbers clustered in the "35.x.y" range [3]. Sonatype's own reporting cautions that this naming pattern is a useful signal for retrospective hunting but not a durable detection mechanism, since "attackers can change names more easily than they can change the purpose of their malware" [3].

The "slop-squatting" framing that appears in early coverage draws on a real and separately documented phenomenon: large language models used for code generation frequently invent package names that do not exist. A 2025 USENIX Security paper by researchers at the University of Texas at San Antonio, the University of Oklahoma, and Virginia Tech generated 576,000 code samples across 16 popular code-generating models and found that 19.7 percent of recommended packages were hallucinated, with open-source models hallucinating at an average rate of 21.7 percent versus 5.2 percent for commercial models, and the worst-performing open models (CodeLlama 7B and 34B) exceeding 33 percent [7]. The same research demonstrated that hallucinated names are frequently reproducible: when 500 hallucination-triggering prompts were re-run ten times each, 43 percent of the fabricated package names reappeared in every single run, meaning an attacker who identifies a commonly hallucinated name and registers it under that identity stands a good chance of it being suggested again to other developers or coding agents [7]. This mechanism — dubbed "slopsquatting" by Python Software Foundation developer-in-residence Seth Larson, by analogy with typosquatting, and independently documented by security researchers tracking the broader trend [6] — was the subject of a CSA AI Safety Initiative research note published in April 2026, which anticipated the general risk of AI-assisted dependency confusion attacks before the Flooding Dropper campaign was reported [8]. Whether Flooding Dropper's specific package names originated from observed LLM hallucinations, from an attacker's own name-generation logic designed to resemble plausible packages, or from some blend of both remains unconfirmed in public reporting, and this note flags that gap rather than resolving it in either direction.

Security Analysis

The technical chain behind Flooding Dropper begins with a departure from the delivery mechanism most supply-chain defenses are tuned to catch. Rather than executing automatically through `preinstall` or `postinstall` npm lifecycle scripts — the pattern that registry-side scanners and many endpoint tools specifically monitor — the malicious packages ship a README that instructs the developer to load the module directly with `require()` [1][3]. This shifts the trigger from an automated install-time hook to a social-engineering step that depends on a human (or an AI coding assistant acting on the developer's behalf) following written instructions, which may explain why some of these packages evaded detection for as long as they did. Once triggered, the code runs a downloader identified as WEL1DROPPER, which fingerprints the host operating system and CPU architecture and requests a matching payload from a set of Cloudflare Workers subdomains, falling back to DNS TXT record delivery from the domain `wel1[.]ru` if the direct HTTPS path is blocked [1]. Platform-specific payload retrieval is further segmented by dedicated subdomains — `sdk.dl.wel1[.]ru` for Linux x64, `ext.dl.wel1[.]ru` for Linux ARM64, `pkg.dl.wel1[.]ru` for macOS, and `net.dl.wel1[.]ru` for Windows — indicating a deliberately engineered, rather than opportunistic, distribution pipeline [1].

The three platform payloads differ meaningfully in capability, which the following table summarizes.

Platform	Evasion/Anti-Analysis	Persistence Mechanism	Final Payload
Windows	Patches Event Tracing for Windows (ETW) and the Antimalware Scan Interface (AMSI); detects sandbox/virtual environments	Registry Run key and a scheduled task	Encrypted binary retrieved as `/pkg/update_win.exe`
macOS	Detects debuggers and analysis artifacts	LaunchAgent	Compiled payload retrieved as `/pkg/beacon_mac.bin`, with DNS TXT fallback
Linux	UPX-packed ELF binary	Auxiliary payloads fetched via	Deploys Sliver, an open-source command-and-control framework

Platform	Evasion/Anti-Analysis	Persistence Mechanism	Final Payload
		Cloudflare Worker	

[1]

The Windows and macOS branches both patch or evade the security telemetry an endpoint would normally rely on to catch this activity, and the Linux branch's choice of Sliver stands out: Sliver is a legitimate, publicly available red-team C2 framework, and this note assesses that its presence on a host is less inherently suspicious than a bespoke malware family's traffic would be, which likely complicates detection through signature-based tooling [1]. Each of the malicious packages also ships a file named `lib/telemetry.js`, which duplicates the downloader's logic but is presented as ordinary usage telemetry — an attempt, as reporting characterizes it, to "add noise and make the malicious behavior look like native profiling" to a developer or automated scanner glancing at the package contents [1]. This decoy pattern echoes the Moika campaign's own use of documentation that described its exfiltration behavior as harmless telemetry, suggesting either shared tradecraft or simple imitation of an approach that worked before [4].

Attribution signals point toward continuity with the Moika campaign's target set rather than a wholly unrelated actor. Analysis of the macOS payload surfaced references to Russian financial institution domains, including `tcsbank[.]ru` and `cloudpayments[.]ru`, echoing Moika's explicit targeting of Sberbank, Alfa-Bank, BCS, and EMCD infrastructure earlier in the year [1][4]. The Hacker News' reporting on this campaign also notes that Palo Alto Networks' Unit 42 has separately documented related npm and PyPI campaigns in 2026 combining cloud credential theft, blockchain-based command-and-control, cryptocurrency wallet theft, and CI/CD credential exfiltration, indicating that the broader pattern of industrialized, high-volume package publishing against developer supply chains extends beyond any single campaign or registry [1]. Taken together, the shift from one or two prolific publisher accounts (as in early Moika waves) to hundreds of small, disposable accounts in Flooding Dropper reads as a plausible adaptation to registry-side takedown and moderation, which is generally more effective against a small number of high-volume publishers than against many low-volume ones — though Sonatype's reporting does not explicitly confirm this was the actor's motivation [3].

Recommendations

Immediate Actions

Organizations should audit developer workstations, build agents, and CI/CD runners for any history of installing packages matching the Flooding Dropper naming patterns identified by Sonatype (terms combining "bigops," "bnpl," and similar strings, with versions in the 35.x.y range), and for any `require()` calls executed against packages obtained in the past several months whose provenance cannot be confirmed [3]. Any host on which an implicated package was actually loaded should be treated as compromised rather than merely "at risk": Sonatype's guidance is to isolate the system, hunt for the specific persistence artifacts described above (Windows Registry Run keys and scheduled tasks, macOS LaunchAgents, or unexplained Sliver C2 traffic on Linux), and rotate developer and CI credentials only after the environment has been fully remediated, since premature credential rotation on a still-compromised host simply hands the new credentials to the same attacker [3]. Security teams should also add the known infrastructure to blocklists at the DNS and proxy layer, specifically the Cloudflare Workers subdomains and the `wel1[.]ru` domain family, including its four platform-specific subdomains [1].

Short-Term Mitigations

Because this campaign specifically bypassed lifecycle-hook monitoring by relying on README-instructed `require()` calls, organizations should extend software composition analysis and endpoint monitoring to flag manual `require()` or `import` statements referencing packages outside a project's committed lockfile, not just packages that execute install scripts [1][8]. CI/CD pipelines should enforce hash-verified lockfiles and block ad hoc installation of packages not already present in that lockfile, a control the CSA AI Safety Initiative's April 2026 slopsquatting research note already recommended as a baseline defense against hallucination-driven package confusion [8]. Given the newness of many of the publisher accounts involved, security teams should also consider flagging or quarantining packages published or first adopted within the prior 30 to 90 days pending manual review, and should specifically restrict any AI coding agent or assistant operating in the environment to an approved package allowlist, routing anything outside that list to a human reviewer before installation [8].

Strategic Considerations

Longer term, organizations that rely on AI coding assistants or autonomous coding agents should treat package-name verification as a mandatory step before any AI-suggested dependency reaches a development or build environment, given the documented tendency of code-generating models to hallucinate plausible-sounding but nonexistent package names at rates as high as 19.7 percent overall and above 33 percent for some open-source models [7]. Procurement and vendor-risk processes for AI coding tools should incorporate a specific question about how the vendor mitigates package hallucination in its generated output, since a vendor's hallucination rate has a direct bearing on its customers' exposure to slopsquatting-style attacks regardless of whether this specific campaign turns out to be hallucination-driven [7][8]. Finally, given that Flooding Dropper appears to be an evolution of the Moika campaign's technique rather than a clean break from it, organizations should expect continued iteration — further waves are likely to adapt naming conventions, delivery mechanisms, or target sectors in response to the detection guidance now public, and threat intelligence monitoring should be tuned to track the underlying behavioral pattern (disposable publisher accounts, platform-specific staged payloads, decoy telemetry files) rather than any single indicator that is likely to change [3][4].

CSA Resource Alignment

This campaign is broadly consistent with the risk model described in CSA's own April 2026 research note, "Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks," which warned that autonomous coding agents installing dependencies without human review checkpoints would create exposure to malicious or fabricated packages. Whether Flooding Dropper's specific names were hallucination-derived remains unconfirmed, as discussed above, but the note's recommended controls — lockfile hash-verification, package-age flagging, and AI-agent allowlisting — apply regardless of that campaign's precise name-generation method, and were published months before this campaign was publicly reported [8]. That note grounded its recommendations in the AI Controls Matrix (AICM) Supply Chain Management domain and in MAESTRO's agentic threat modeling framework, both of which remain the most directly applicable CSA frameworks for organizations assessing exposure to this incident; the AICM's current version, AICM v1.1, addresses provenance tracking and software bill of materials (SBOM) requirements that map to the immediate actions recommended above [8][9].

This is also one of several 2026 npm supply chain campaigns to deliver a cross-platform RAT into developer environments, following CSA's coverage of Sapphire Sleet's compromise of the Mastra AI npm ecosystem, in which a hijacked maintainer account was used to poison 145 packages in 88 minutes, and CSA's broader whitepaper "npm Supply Chain Under Siege: TeamPCP, Miasma, and npm v12," which documents how AI-adjacent tooling packages that aggregate credentials across multiple LLM providers

create disproportionate blast radius when compromised [10][11]. Organizations that used CSA's guidance from those reports to harden npm scope and maintainer-account monitoring should extend that same monitoring to the disposable-account, high-volume-publishing pattern Flooding Dropper introduces, since the underlying lesson — that publisher identity and account age are an important complement to package-content signals — applies across all of these incidents.

References

[1] The Hacker News. "Nearly 800 Malicious npm Packages Deliver Cross-Platform RAT and Infostealer." The Hacker News, August 2026.

[2] SC World. "Nearly 800 malicious npm packages deliver cross-platform malware." SC World, August 2026.

[3] Sonatype. "Flooding Dropper Hits npm With 850 Malicious Packages." Sonatype Blog, August 2026.

[4] SafeDep. "183 npm Packages Target Cloud and Finance via oob.moika.tech." SafeDep Threat Intelligence, 2026.

[5] Security Boulevard. "Flooding Dropper Hits npm With 850 Malicious Packages." Security Boulevard, August 2026.

[6] Socket. "The Rise of Slopsquatting: How AI Hallucinations Are Fueling a New Class of Supply Chain Attacks." Socket Blog, 2026.

[7] Spracklen, J., Wijewickrama, R., Sakib, A.H.M.N., Maiti, A., Viswanath, B., and Jadliwala, M. "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs." 34th USENIX Security Symposium, 2025.

[8] Cloud Security Alliance AI Safety Initiative. "Slopsquatting: AI Code Hallucinations Fuel Supply Chain Attacks." CSA Research Note, April 2026.

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

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

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

