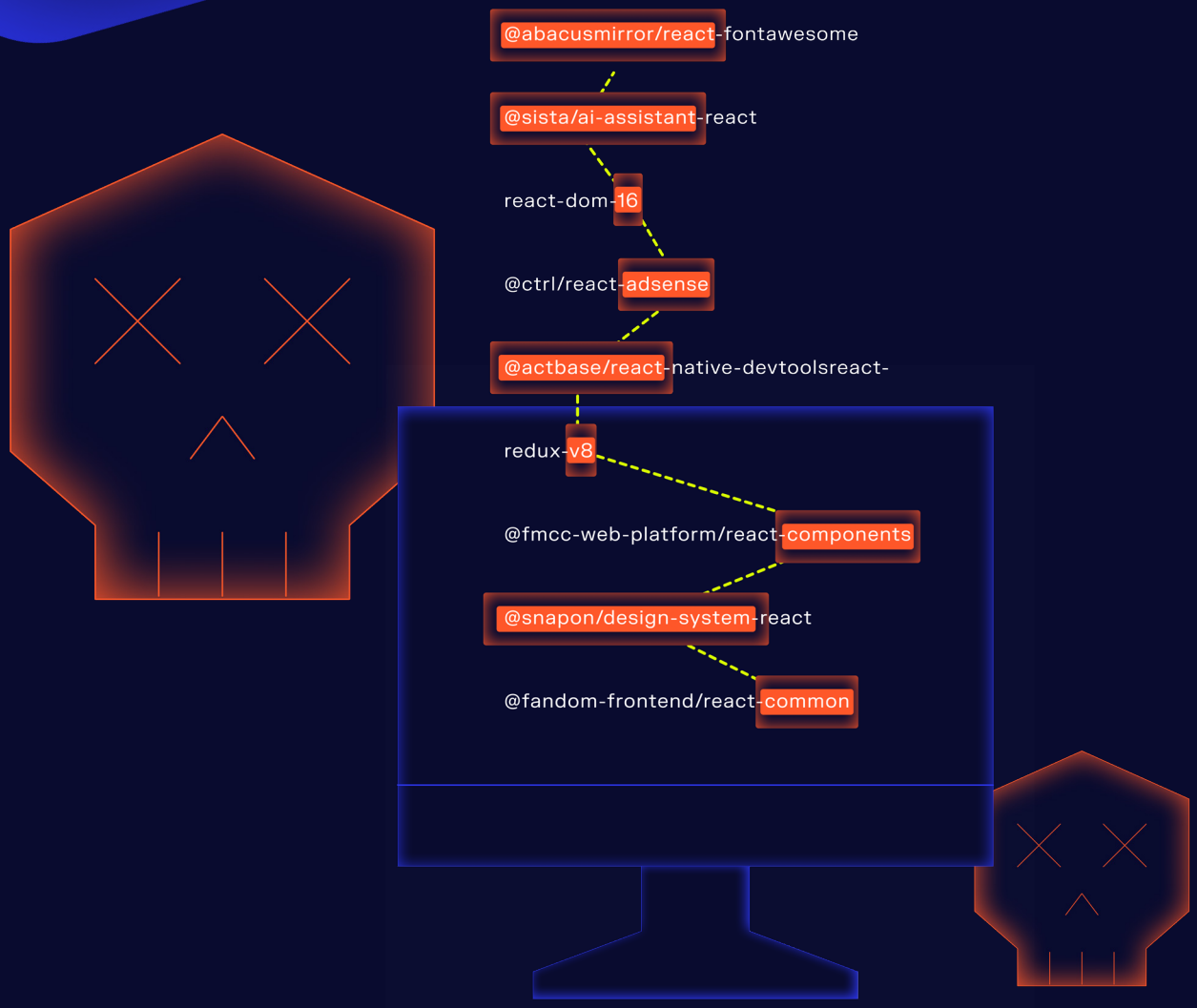
sonatype

BEYOND TYPOS

Threat Actors Using Naming-Variants to Steal Developer Data



Introduction

Open source package attacks have outgrown the classic typosquatting attack story. In this dataset of 4,309 malicious packages, attackers rarely relied on obvious misspellings alone. Instead, they used names that looked believable in the flow of real development work: plugins, configs, SDKs, utilities, scoped modules, wrappers, and versioned variants.

According to Sonatype analysis, 91% of brandjacking open source malware went beyond the common typosquat vector. These packages succeeded not because they were visually identical to trusted projects, but because they borrowed the language and structure of legitimate software ecosystems.

That distinction matters because these packages are not harmless lookalikes. The most common malicious behaviors were host information exfiltration, secrets exfiltration, droppers, and backdoors — turning a routine install into a path for reconnaissance, credential theft, and follow-on compromise.

"Typosquatting" is now too narrow a label for what this analysis captures. The broader pattern is manufactured legitimacy: attackers designing package names to look plausible, useful, and operationally routine inside modern software ecosystems.

> **ATTACKERS ARE EXPLOITING NAMING CONVENTIONS AND WORKFLOW FAMILIARITY, NOT JUST HURRIED KEYSTROKES.**

91%

OF MALWARE WENT BEYOND THE TRADITIONAL TYPOSQUAT ATTACK VECTOR

147

CAMPAIGN FAMILIES OBSERVED, SIGNALING INDUSTRIALIZATION

74%

OF MALICIOUS PACKAGES TARGETED DEVELOPER DATA — ENVIRONMENTS, SECRETS, OR BOTH

540

MALICIOUS PACKAGES TARGETED THE REACT FRAMEWORK

Attackers are Manufacturing Legitimacy to Abuse Developer Trust

Brandjacking attacks, or naming-variant attacks, work by making malicious packages appear connected to trusted open source projects. But the data shows that attackers no longer simply rely on misspellings, commonly known as typosquats. They use complex naming structures that make malicious packages look like plausible parts of real software ecosystems.

FIGURE: 1

Types of Brandjacking Attacks

Brandjack Subtype	Example	Definition
Typosquat	**lodsh** instead of **lodash**	Exploits common misspellings.
Suffix Addition	**react-plugin**	Adds words like “plugin” or “utils.”
Prefix Addition	**auth-express**	Prepends terms like “auth” or “secure”
Embedding	**uxcamreactexample**	Hides inside a familiar word.
Dependency Confusion	**@company/internal**	Mimics internal package names.
Version Mimicry	**axios-v1.2.3**	Copies popular version numbers.

Naming-variant attacks go further. They use structural or semantic changes that make a package appear adjacent to a trusted project rather than identical to it.

WHAT IS TYPOSQUATTING?

A typical typosquat imitates a trusted package through a small spelling variation, often relying on a developer mistyping a name.

These changes can include suffixes, prefixes, embedded target terms, altered scopes or namespaces, dependency-confusion patterns, version-like markers, or names that resemble the function of a legitimate package.

THE DIFFERENCE IS IMPORTANT:

CLASSIC TYPOSQUATTING DEPENDS ON ERROR. NAMING-VARIANT ATTACKS DEPEND ON PLAUSIBILITY.

Naming Variants Are the Dominant Pattern

In this dataset, attackers overwhelmingly favored naming-variant tactics over traditional typosquatting. 91% of confirmed malicious packages were naming-variant attacks, compared with 9% typosquats. That means most malicious package names were not designed to look like obvious mistakes. They were designed to look like ordinary extensions, helpers, integrations, or ecosystem add-ons.

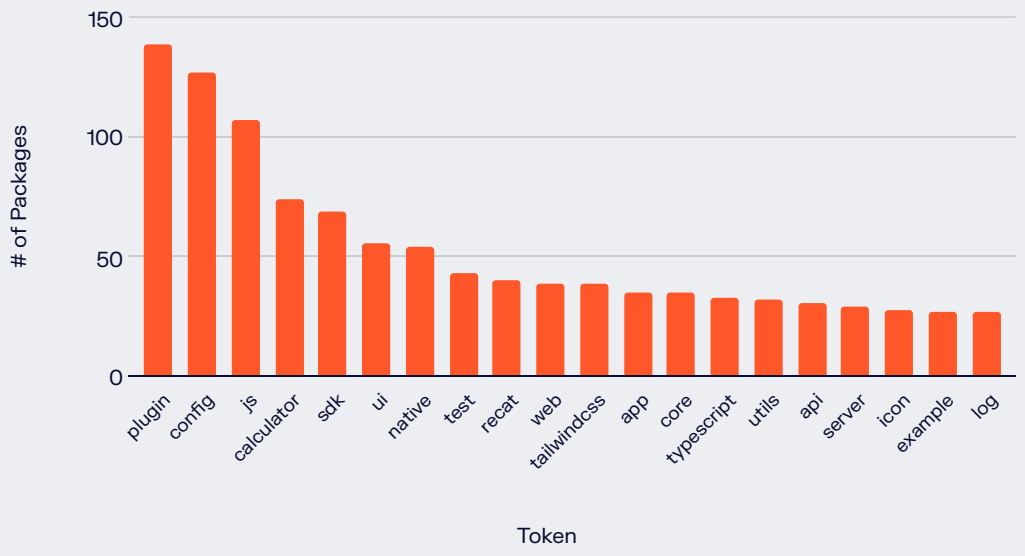
Why These Names Work

These names work because they feel contextually plausible. Developers expect popular frameworks and tools to have surrounding ecosystems of plugins, configs, SDKs, wrappers, utilities, scoped modules, and versioned variants. Attackers abuse trust by borrowing the structure and vocabulary of legitimate development work, not just the spelling of trusted packages.

Terms like plugin, config, and sdk can feel routine because those packages often integrate with frameworks, tools, build processes, or runtime environments. That expectation gives attackers more room to hide data theft, droppers, or other multi-stage behaviors without immediately triggering suspicion.

FIGURE: 2

Top 20 Tokens Added to Malicious Package Names



What the Naming Patterns Reveal

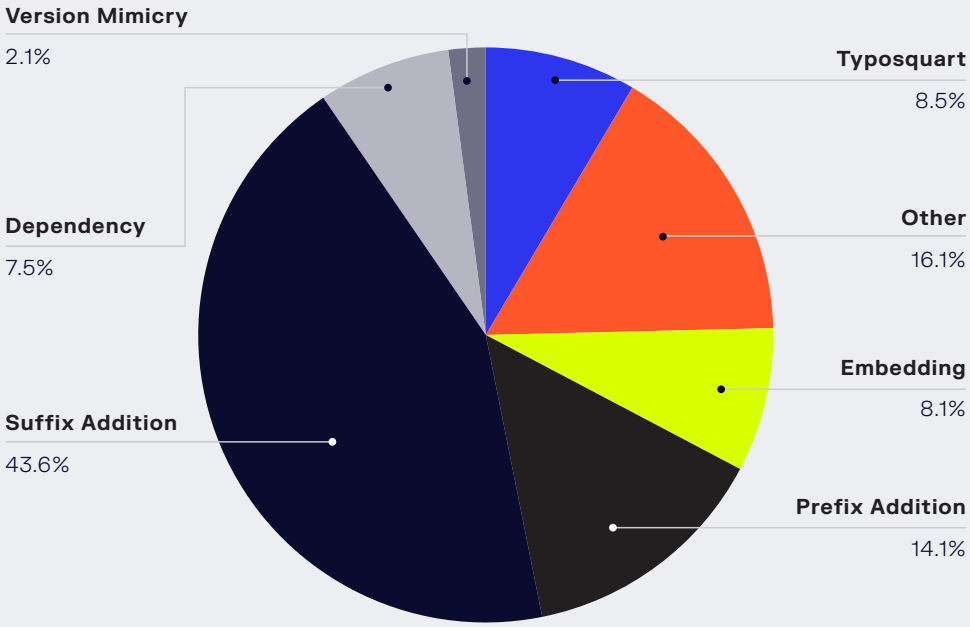
The structural changes are not random. Suffix addition was the most common naming variant, accounting for 43.6% of all observed tactics. It was also tied to major malicious objectives, including 892 instances of secrets exfiltration and 612 droppers.

The main patterns include suffix addition, prefix addition, embedding, dependency confusion, version mimicry, and semantic resemblance. Each makes a package appear connected to a legitimate project without needing to be a direct misspelling.

That is why “typosquatting” is too narrow a label for the broader pattern this analysis captures. Attackers are designing package names that look normal in the places developers already expect complexity.

FIGURE: 3

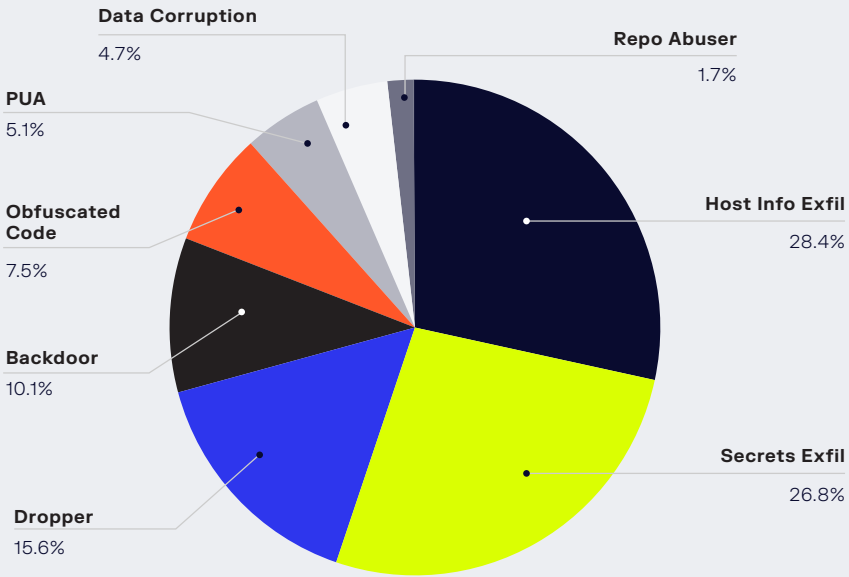
Breakdown of Packages by Attack Vector Subclass



The Objectives: Data Theft and Multi-Stage Infections

FIGURE: 4

Brandjacking Attacks by Threat Type



Brandjacking attacks turn a plausible package name into execution, access, and follow-on compromise. The threat-type distribution shows a clear pattern: the largest behaviors are host information exfiltration and secrets exfiltration, followed by droppers, backdoors, and obfuscated code. In other words, attackers are using trusted-looking package names to steal developer data, stage later payloads, and preserve access.

Note: Threat labels are non-exclusive; one package may contribute to multiple categories. Percentages reflect the distribution of recorded threat labels across confirmed brandjacking packages.

Data Theft Creates Leverage
Host information and secrets exfiltration account for more than half of observed threat labels. That means attackers are prioritizing the data most likely to unlock broader access: environment variables, API keys, package registry tokens, GitHub tokens, cloud secrets, SSH credentials, and CI/CD secrets.

Those credentials can amplify the attack beyond the original install. They can be used to access source code, publish malicious updates, poison build systems, reach cloud infrastructure, or move deeper into internal environments. In downstream breach scenarios, stolen developer or CI/CD credentials can turn one compromised package into a path toward trusted software projects, customers, or internal systems. In self-propagating attacks, stolen package manager tokens can help the campaign spread through the same ecosystem trust that made the first install possible.

Multi-Stage Attacks Extend the Compromise
Droppers, backdoors, and obfuscated code show that many attacks are built to continue after installation. The first package may simply create the opening: fetching a second payload, enabling remote access, or hiding malicious behavior long enough to evade review.

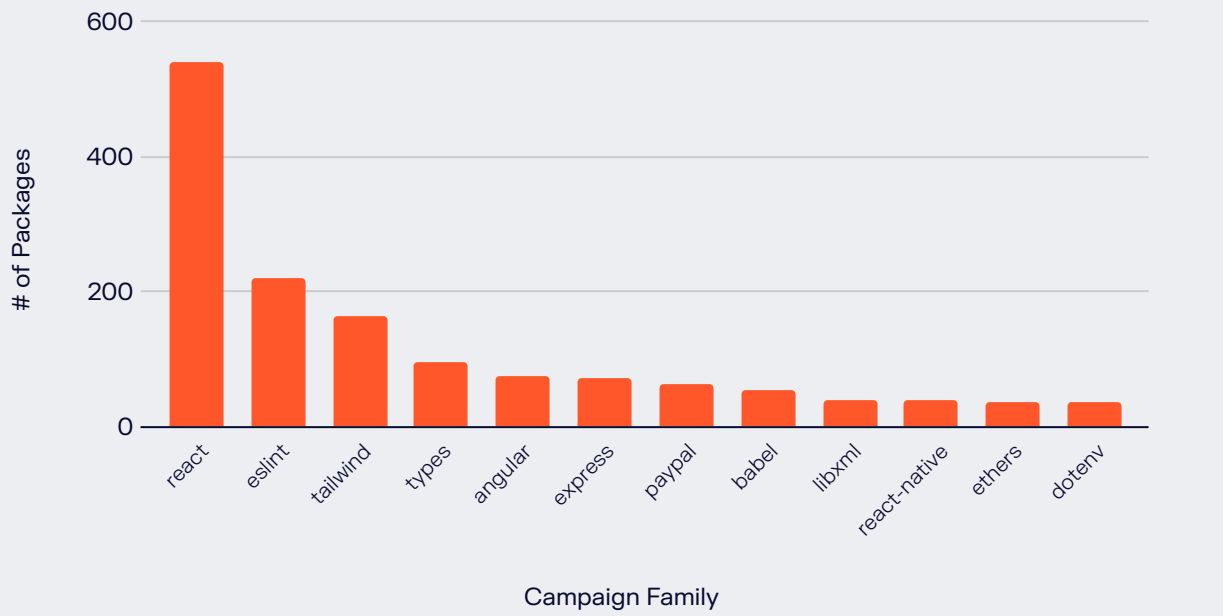
The near absence of cryptomining is telling. Attackers are no longer focused on stealing compute; the goal is to steal the access paths that let them expand.

Attackers Target Ecosystems Where Trust Is Already Distributed

Brandjacking activity is not spread evenly across the open source ecosystem. It clusters around high-trust, high-usage package families where adjacent packages are already common.

FIGURE: 5

Breakdown of Campaign Families By Targeted Ecosystems



That concentration is not accidental. These ecosystems all have naming environments where new or unfamiliar packages can seem legitimate at a glance. React has a broad universe of UI components, wrappers, utilities, and React Native-related packages. ESLint has a mature plugin and config ecosystem. Tailwind has common add-ons for forms, typography, animation, configuration, and styling. Crypto and DeFi tooling often depends on integrations with wallets, exchanges, smart contract tooling, and blockchain libraries.

FIGURE: 6

Top Open Source Ecosystems Targeted by Brandjacking Malware

Ecosystem	Malicious Packages	Legitimate Packages Targeted	Naming Context	Dominant Threat Behaviors
React	540	77	UI components, utilities, wrappers, React Native-related packages	Data exfiltration, droppers, obfuscated code
ESLint	220	36	Config and plugin ecosystem	Secrets exfiltration, droppers
Tailwind	165	13	Forms, typography, animation, config, styling	Data exfiltration, droppers
Crypto/DeFi	114	10	Ethers, coinbase, hardhat integrations	Secrets exfiltration, credential theft, data exfiltration

The common denominator is package adjacency. Attackers appear to favor ecosystems where trust is already distributed across many names, maintainers, extensions, and add-ons. In those environments, legitimacy is easier to imitate because developers already expect a long tail of helpers, wrappers, plugins, configs, and integrations.

This makes brandjacking especially effective in modular ecosystems. A malicious package does not need to look identical to a trusted package. It only needs to look like it belongs nearby.

FIGURE: 7

Quarterly Heat Map: Top Campaign Families by # of Packages

Quarter	Q2 2024	Q3 2024	Q4 2024	Q1 2025	Q2 2025	Q3 2025	Q4 2025	Q1 2026
react	20	27	**61**	**60**	**77**	**133**	**118**	**78**
eslint	12	20	51	25	19	59	27	18
tailwind	0	1	1	2	3	40	**88**	31

Industrialized Cyberattacks Have Overtaken Open Source

These attacks do not look like isolated, one-off attempts. They cluster into repeatable campaign families, pointing to structured playbooks rather than random malicious publishing. The pattern suggests malicious package abuse is being executed systematically, with attackers reusing recognizable methods instead of improvising from package to package.

The clearest signal is cross-family reuse. Attackers appear to operate across multiple package families, reusing infrastructure, identities, and naming tactics instead of staying confined to a single ecosystem. That makes the threat look industrialized, not opportunistic: the same patterns show up repeatedly across environments where similar trust dynamics exist.

FIGURE: 8

Publisher Behavior Shows Brandjacking Is Not One Uniform Threat

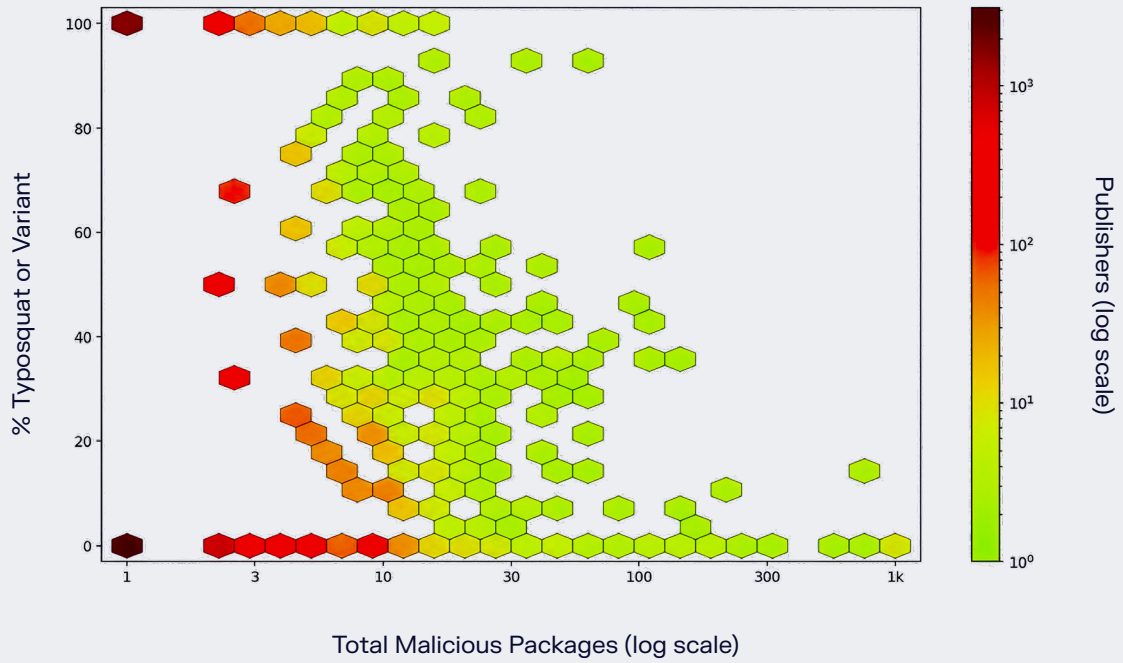


Figure 7: Hexagonal density map of malicious publishers. Each hex represents a cluster of publishers with similar package volume (x-axis, log scale) and Typosquatting rate (y-axis). Color intensity shows publisher count (log scale). Two distinct populations emerge: publishers that exclusively publish typo squats (top edge) and those that never do (bottom edge), with a smaller mixed population in between.

Publisher behavior reinforces that conclusion. Sonatype observed distinct attacker modes, with some publishers concentrating on typosquatting, some avoiding it entirely, and others mixing tactics. That suggests multiple playbooks, not one uniform threat model. Scale matters too: some publishers are associated with very large volumes of malicious packages, often concentrated in naming-variant behavior rather than classic typo-only approaches.

The larger takeaway is that package impersonation is becoming more organized and repeatable. Defenders will miss the real pattern if they evaluate suspicious packages one at a time. The better lens is the campaign, the publisher cluster, and the repeated behaviors that connect activity across ecosystems.

Why Do Current Defenses Miss This?

Traditional defenses are increasingly missing the wormable malware making headlines because many controls still assume the threat will look like obvious typosquatting attacks. As soon as a sneakier package slips through, the organization is compromised.

Most engineering and DevOps defenses were built to catch known-bad packages, explicit policy violations, or clear misspellings. That model is weaker against packages that look operationally normal: a new plugin, config helper, scoped module, SDK, wrapper, or versioned variant that appears consistent with the surrounding ecosystem.

THE BLIND SPOT APPEARS AT THE MOMENT OF ADOPTION.

When a developer adds a dependency, updates a lockfile, installs a helper package, or pulls in something framework-adjacent, a plausibly named package may not trigger review — especially if it is new and not yet widely flagged. Reputation-based controls also have a timing problem. By the time a suspicious package has enough external signals to look risky, it may already have reached test pipelines, CI jobs, ephemeral environments, or developer workstations.

Many teams assess dependencies one package at a time, while attacker behavior increasingly looks campaign-based. A package that seems low-signal in isolation may look far more suspicious when viewed alongside publisher behavior, naming patterns, and activity across multiple ecosystems.

What Defenders Should Do Differently: A Checklist

This is less a problem of developer carelessness than control design. If your process assumes malicious packages will look obviously wrong, or that reputation will mature before adoption, you are defending against a weaker threat model than the one reflected in this dataset.

- **Don't treat plausible names as proof of safety:** Modern brandjacking often borrows legitimate ecosystem naming conventions, so defenses should evaluate package context, publisher behavior, and campaign patterns — not just whether a name looks suspicious.
- **Add friction for new or first-seen dependencies:** Require extra review before unfamiliar packages enter the build, CI pipeline, or developer environment.
- **Scrutinize packages that look ecosystem-adjacent:** Pay closer attention to packages that appear to be plugins, configs, SDKs, helpers, wrappers, utilities, or scoped variants of trusted projects.
- **Review names for contextual plausibility:** Look for suffix additions, prefix additions, scoped variants, dependency-confusion patterns, version mimicry, semantic resemblance, and embedded target terms.
- **Monitor publisher behavior, not just package behavior:** Assess whether publishers are connected to repeated naming tactics, multiple package families, or suspicious campaign-style activity.
- **Evaluate risk at the campaign level:** Suspicious patterns can become clearer when viewed across campaign families, publisher clusters, and repeated behaviors.
- **Apply stronger controls in adjacency-heavy ecosystems:** Increase scrutiny in modular ecosystems like React, ESLint, and Tailwind, where unofficial plugins, configs, helpers, and extensions are common.
- **Create meaningful pause points before adoption:** The goal is to slow down risky dependency introduction long enough to verify whether a plausible-looking package actually belongs.

Beyond Typosquatting Means Beyond Typo Detection

The central lesson from this analysis is that open source package abuse has moved beyond the classic typosquatting story. The next generation of attacks is designed to look plausible, useful, and operationally normal within modern software ecosystems.

Attackers are not succeeding only because developers mistype package names. They are succeeding because malicious packages increasingly borrow the language, structure, and extension logic of legitimate development work — especially in ecosystems where plugins, configs, wrappers, and helpers are everywhere.

That changes the defensive challenge. The problem is no longer just spotting names that look wrong; it is recognizing when a package looks normal for exactly the wrong reasons.

For engineering and security leaders, the implication is straightforward: defending against modern package abuse requires more than typo detection or static reputation checks. It requires evaluating naming patterns, publisher behavior, campaign context, and the workflows through which new dependencies enter the environment.

> **THE NEXT GENERATION OF OPEN SOURCE PACKAGE ABUSE IS DESIGNED TO LOOK NORMAL.**