ManageEngine
Endpoint Central

Open source supply chain attacks

Why continuous telemetry, not periodic scans, ***closes the gap***

www.manageengine.com/endpoint-central

Why this is surfacing now

Supply chain attacks are not a new idea. The world first learned to take them seriously after **SolarWinds**, the incident that proved a single compromised dependency could cascade into catastrophic damage across thousands of organizations at once. That case made one thing evident: any architecture built on shared, trusted code hands an attacker an easy, hidden path inside. Once that path is compromised, the damage does not stay contained to one victim. It spreads to everyone downstream who depended on that code without knowing it.

> **What has changed since then is not the concept. It is the pace, and who is now running the playbook.**

The pattern tends to start in the same place every time. The actors with the most patience, the deepest resources, and the backing of a state move first. These are the **advanced persistent threats, the APTs**: the only groups with the time, funding, people, and staying power to run an operation for years before it pays off. Once an APT proves a technique works, smaller, financially motivated groups pick it up and scale it. This is the same progression ransomware went through: a method pioneered by a well resourced, patient actor becomes, within a few years, the default playbook for everyone else. Open source supply chain compromise is now walking that same road. Treat it as the opening move, the one other, less sophisticated attackers will spend the next few years copying.

The maintainer's blind spot

To understand why the open source ecosystem is such fertile ground, start with the people who keep it running. Open source maintainers are not paid for this work. They do it in their own time, on top of everything else in their lives, and like anyone, they have a limit to how much they can absorb in a day.

Even in ordinary times, a popular project receives a steady stream of pull requests. Since AI coding tools became common, that stream has turned into a **flood**, and most of it is genuine, well intentioned code. Somewhere inside that flood, an attacker slips in a single malicious change dressed up as a routine, one line fix. A tired maintainer, worn down by the sheer volume of legitimate requests and short on the time a careful review demands, waves it through. That gap, between the energy proper review requires and the energy an overloaded maintainer has left to give, is exactly the opening an attacker is waiting for.

Two ways in

01 **The first method targets packages that are popular on the surface but forgotten underneath.** Take any widely used package with strong, attentive maintainers. Somewhere inside its dependency tree usually sits a much smaller library that nobody has touched in years, one that everyone still quietly relies on but nobody actively maintains anymore. Attackers take control of that abandoned piece. Because the popular, trusted package still depends on it, the compromise rides in on the back of code that thousands of teams already trust without a second look.

02 **The second method exploits the rise of AI assisted coding itself.** npm, the Node.js package registry, is the dominant platform for modern development, and AI coding models increasingly suggest packages to developers by name as they write code. The trouble is, models sometimes hallucinate: they invent a plausible sounding package name that does not actually exist. Picture a name like **structured logging**. A developer working with AI generated code sees that name recommended and simply installs it, assuming it is real. Attackers watch for exactly these hallucinated names, publish an actual package under that exact title, and wait. Every developer who lets the suggestion install unchecked ends up pulling the attacker's package straight onto their own machine.

> **Both routes point to the same conclusion. This is no longer a rare, isolated technique. Attackers have made the open source supply chain a primary way in, and they are working it from more than one direction at once.**

When it becomes real: a case that shows the scale

The next question is scale. Has an attack of this kind ever caused damage on the order of a major ransomware outbreak? One recent case answers that clearly.

Its impact traced back to one of the most heavily used utility packages in the JavaScript ecosystem, **axios** among the packages caught up in the campaign. The group behind it was organized and fast: they were **compromising a package roughly every two days**. Through these packages, they harvested a trove of exposed data sitting inside build environments: secret keys, client tokens, the credentials that let applications and services talk to each other and to the infrastructure behind them.

The first level of damage looked contained: a set of developers had their machines affected, nothing more. The second level is where it turns serious. The attackers did not just collect that trove of credentials and walk away. They parsed it, validated it piece by piece, and used it to push further in. In one case, an organization discovered its cloud environment had been breached. Tracing the path back led to a single stolen token, and the token traced back to one developer machine, where that same compromised package had quietly exfiltrated a file containing the credential. That exploitation ran for roughly **four months**, and even now, the assumption is that the attackers are still mining what they took, still working out everything it can be used for.

That is the picture as far as anyone can confirm publicly. But whenever an attack like this comes to light, the only safe assumption is that more is happening somewhere still invisible to us. **Assume the worst until it is proven otherwise**: assume a credential exposed in one incident is already being used to build toward a larger compromise somewhere not yet found. Nobody can prove that is not happening right now, in some environment, to someone. That is the posture to take.

What this case demonstrates is the real danger of the open source supply chain. It is not the size of any single breach. It is the door that one compromise opens onto everything downstream. A supply chain compromise rarely stays contained. It becomes the entry point for the next, larger one. That is what makes it critical, and that criticality is exactly what has to be addressed.

The Linux backdoor: three years for one day

Few cases illustrate just how patient, and how deliberate, this kind of operation can be as clearly as the Linux story.

The package at the center of it sat at the very core of Linux, bundled into the operating system itself: a compression utility called **XZ Utils**, tracked today as **CVE-2024-3094**. Behind the compromise was a developer identity that, as far as anyone can tell, never existed as a real person: a fabricated persona later reporting refers to as **Jia Tan**.

The account first appeared on GitHub in **January 2021**. Its first known contribution to the XZ Utils project came in October of that year. From there, over roughly **three years**, Jia Tan built a reputation the ordinary way: steady, useful contributions, the kind that earn a maintainer's confidence bit by bit, until the real maintainer of the library came to rely on this contributor and, eventually, handed over **genuine commit access**.

The moment that access arrived, the persona moved. Three years of quiet, careful trust building turned, almost immediately, into an attempt to insert a backdoor into one of the most widely distributed pieces of software on earth.

It was caught, but only by chance. A single developer, checking something unrelated, noticed the package behaving strangely and traced it back to the source. The backdoored version had been live for exactly one day before it was pulled.

> **That is the detail worth sitting with. Three years to build trust. One day live before discovery. Had that one developer not noticed, this backdoor would have shipped inside major Linux distributions worldwide.**

This is what an operation with real patience, real resources, and no pressure to rush looks like, and it is exactly why it is classified, in hindsight, as **advanced persistent threat activity**: the kind of operation only an actor with years to spend and nothing forcing their hand could ever run.

Two different attacks, one shared lesson

The axios linked breach shows how a single stolen credential in a developer's build environment can become the opening move in a much larger compromise, and how long that exploitation can run before anyone sees the full shape of it: four months, in that case. The Linux backdoor shows how much patience a sophisticated actor is willing to spend for a single moment of access, and how close that moment came to reaching every major Linux distribution in the world: three years of trust building, undone or discovered in one day.

Open source code is not a side concern sitting outside the walls of an organization's own network. It is **already inside**, compiled into the applications teams run every day, trusted at the same level as code written in house. Securing the endpoints and the network is no longer enough on its own. The supply chain that builds the software running on those endpoints has become an attack surface in its own right, and it needs to be watched with the same seriousness. Which raises the real question: watched how.

Why continuous telemetry, not SBOM

The standard answer the industry has reached for is the software bill of materials, **SBOM**: a full inventory of what is in your software, the ingredients label for every application you run. SBOM is valuable, and no one should skip it. But it is a point in time photograph, not a live view.

If a package is compromised after the last SBOM scan, there is no visibility into that. If a developer installs a new dependency this morning that was poisoned only last night, the SBOM does not know about it yet. When a security operations center gets an incident alert and pulls up SBOM data to investigate, what they are looking at is a **stale inventory, not a live trail** they can follow.

SBOM

A point in time photograph

A full inventory of what is in your software, the ingredients label for every application you run.

CONTINUOUS TELEMETRY

A continuously running camera

A full, searchable audit trail that never stops running.

Look back at both stories above. The Linux backdoor was live for a single day before a developer happened to notice. The axios linked breach ran for roughly four months before its full shape became visible, and even now nobody can say the exploitation has stopped. A snapshot taken once, at a fixed point in time, would have missed both of those windows completely. What both cases needed was not a better photograph. It was a continuously running camera.

Closing the gap: what continuous telemetry looks like in practice

Continuous package telemetry.

Package manager activity, npm, pip, and the rest, is tracked as a first class signal, not an afterthought. Every install, every update, every dependency resolution, every version change is recorded with a timestamp, a user, and a machine identity attached. Not a snapshot taken once a quarter. A full, searchable audit trail that never stops running.

Central threat intelligence on compromised packages.

The intelligence layer behind this is continuously updated with indicators tied to known malicious package versions, the same way new indicators are pushed out for malware hashes or command and control domains. The moment a poisoned package is flagged anywhere in the wild, that intelligence propagates automatically to every deployment watching for it.

Live investigation when an incident triggers.

An analyst can search by package name and version and get back exactly when it was installed, by whom, on which machine, what it pulled in with it, and whether any of that is now flagged as malicious: the entire dependency chain, historically, for any endpoint across the fleet. That is the difference between a stale inventory and a live trail.

Hold and quarantine before the damage lands.

If a package install matches a compromised indicator, the action is held and the security team is alerted immediately. Once the threat is confirmed, the affected machine can be quarantined before the malicious code ever executes or exfiltrates anything.

Closing thought

The same lesson keeps repeating across the security stack: the very tools built to protect an endpoint can, in the wrong hands, be turned into a weapon. Open source supply chain attacks are that lesson showing up at a different layer of the stack: the tools developers trust to build software are now being weaponized too.

Regulatory mandates are already clear on patching and vulnerability management. But the frontier has moved past that. The question is no longer only which known vulnerabilities remain unpatched. It is **which packages were installed, when, by whom, and whether they were clean at the moment they ran**. That is not a job a periodic SBOM scan was ever built to do. It is a job for continuous telemetry, live intelligence, and a detection layer that treats package managers as the attack surface they have quietly become.

Validated by experts. Championed by users

Gartner Peer Insights™

90% of Gartner Peer Insights™ 'Voice of the Customer' Reviewers recommend Endpoint Central

Gartner

A Challenger in the 2026 Gartner Magic Quadrant for Endpoint Management Tools

IDC

Leader in the IDC MarketScape: Worldwide Unified Endpoint Management Software 2025–2026 Vendor Assessment

AV comparatives

97.4% Protection Rate, with low impact on system performance

Next steps

Bring the package managers your developers actually use — npm, pip, and the rest. We will show you the install trail behind any endpoint in your fleet, and exactly where a compromised version would be held.

Or simply sign up for a free demo:

www.manageengine.com/products/desktop-central/request-demo.html

REQUEST A FREE DEMO

ManageEngine Endpoint Central