

Newly Discovered durabletask Malware Targeted Kubernetes, Cloud Secrets, and CI/CD Infrastructure

 upwind.io/feed/malicious-node-ipc-npm-package-credential-theft

Morgan Pearson

May 14, 2026



Executive Summary

On May 14, 2026, malicious versions of the widely used `node-ipc` npm package were published through a legitimate maintainer account, introducing a sophisticated credential-stealing payload into a package with approximately 3.35 million monthly downloads. The malicious payload was hidden inside the CommonJS bundle (`node-ipc.cjs`) and silently executed whenever applications loaded the package through `require('node-ipc')`.

The malware harvested developer, CI/CD, cloud, Kubernetes, SSH, and AI tooling credentials before exfiltrating the collected data through DNS TXT queries to attacker-controlled infrastructure. Unlike many traditional malicious npm packages, the malware was specifically engineered to avoid disrupting normal application behavior while operating silently in the background.

This campaign demonstrates the continued evolution of software supply chain attacks from simple downstream malware delivery into infrastructure-aware credential harvesting

operations targeting CI/CD pipelines, cloud identities, and developer environments.

At the time of analysis, the command-and-control infrastructure remained live.

Attack Timeline

Time (UTC)	Event
2026-05-14 07:26	Domain azurestaticprovider[.]net registered via NICENIC
2026-05-14 ~14:25	node-ipc@9.1.6, 9.2.3, 12.0.1 published simultaneously
2026-05-14 17:45	Upwind Supply Chain Detector triggered on 12.0.1
2026-05-14 18:07	C2 IP 37.16.75[.]69 confirmed live and resolving

The package had remained untouched for approximately 20 months before the coordinated malicious publication campaign.

Affected Versions

The following versions were confirmed malicious:

Package	Status
node-ipc@9.1.6	Malicious
node-ipc@9.2.3	Malicious
node-ipc@12.0.1	Malicious

All three versions contained the exact same obfuscated payload appended to node-ipc.cjs.

What Happened

The attackers modified the package structure so that CommonJS consumers automatically loaded a malicious bundle:

```
"main": "node-ipc.cjs",
"exports": {
  "import": "./node-ipc.js",
  "require": "./node-ipc.cjs"
```

```
}
```

Copied

This change caused all CommonJS consumers of `require('node-ipc')` to silently execute the malicious bundle while ESM imports continued appearing legitimate.

The malicious payload was appended directly after the legitimate `module.exports` section inside the generated bundle:

```
// Legitimate code ends here:
0 && (module.exports = {
  IPCModule
});

// Malicious payload begins:
(function(_0xaed59b,_0x282d65){...
```

Copied

The package remained fully functional while malicious activity executed silently in the background, significantly reducing the likelihood of immediate detection.

How Was node-ipc Compromised?

New evidence strongly suggests the node-ipc compromise originated from an expired maintainer email domain takeover rather than malware, CI compromise, or direct npm infrastructure abuse.

The malicious packages were published through the legitimate npm maintainer account:

atiertant

belonging to French software developer Alexandre Tiertant, a long-standing contributor associated with the node-ipc ecosystem and multiple legitimate open-source projects.

The account was registered using the email address:

a.tiertant@atlantis-software.net

Research indicates the domain `atlantis-software.net` expired and was re-registered by the attacker on May 7, 2026 through Namecheap. Shortly afterward, the attacker configured mail infrastructure using Namecheap PrivateEmail, effectively gaining control of the original maintainer's inbox.

With access to the email address, the attacker could simply use npm's legitimate password recovery workflow:

- trigger "Forgot password,"
- receive the reset email,
- reset the password,
- and fully take over the maintainer account.

No npm systems needed to be breached. No malware or credential theft was required. Control of the expired email domain alone was enough to inherit ownership of the maintainer identity.

The attacker then appeared to spend several days preparing the operation:

- verifying publish access to `node-ipc`,
- building the malicious `node-ipc.cjs` payload locally,
- and registering the command-and-control domain `azurestaticprovider.net` only hours before the malicious publication campaign.

At approximately 14:25 UTC on May 14, 2026, the attacker published:

- `node-ipc@9.1.6`
- `node-ipc@9.2.3`
- `node-ipc@12.0.1`

All three versions contained the identical malicious payload.

Notably, there were:

- no corresponding Git tags,
- no matching GitHub commits,
- and the malicious `node-ipc.cjs` bundle did not exist anywhere in the public repository.

This strongly suggests the malicious packages were built and published locally after direct npm account takeover rather than through the project's normal source control workflow.

The incident highlights a broader security gap affecting many developer ecosystems: once an email domain expires, any attacker who re-registers it may inherit password reset access to developer accounts tied to that domain, including npm, GitHub, PyPI, cloud platforms, CI/CD systems, and other identity providers relying on email-based account recovery.

Technical Breakdown

Stage 1 – Silent Execution

The malware intentionally avoided interrupting the requiring application.

When loaded through:

```
require('node-ipc')
```

the payload deferred execution asynchronously:

```
setImmediate(function() {  
  void main().catch(() => { process.exitCode = 1; });  
});
```

Copied

This allowed malicious activity to execute one event-loop tick later while the application itself continued functioning normally.

Stage 2 – Sensitive API Initialization

The malware immediately imported a large number of sensitive Node.js APIs:

```
var fs          = require("fs");  
var crypto      = require("crypto");  
var dns         = require("dns");  
var childProcess = require("child_process");
```

Copied

These APIs enabled:

- credential harvesting
- archive construction
- DNS exfiltration
- process spawning
- filesystem access

The payload also collected:

- OS details
- hostnames
- environment variables

- architecture information
- network topology
- runtime metadata.

All environment variables were dumped directly from the process environment:

`Object.keys(process.env)`

which may expose:

- cloud credentials,
- CI/CD secrets,
- API keys,
- authentication tokens.

Stage 3 – Credential Harvesting

The malware targeted an extensive set of credential and secret locations, including:

Cloud Credentials

- AWS
- Azure
- GCP
- OCI
- Alibaba Cloud

CI/CD and Infrastructure

- GitHub Actions
- GitLab runners
- Terraform
- Kubernetes
- Docker

Developer Secrets

- SSH keys
- .env files
- shell history
- npm credentials
- browser profiles

AI Tooling

- Claude configs
- Kiro configs

The malware contained 114 path patterns specifically designed to locate infrastructure secrets and developer credentials.

Stage 4 – Archive Construction and Encryption

Collected files were packaged into a custom tar archive built directly in memory using handcrafted ustar headers:

```
Buffer.from("ustar\0", "utf8").copy(header, 257);
```

The archive was then:

1. compressed using gzip,
2. encrypted using a custom XOR/HMAC-based scheme,
3. chunked into small payloads,
4. prepared for DNS exfiltration.

Stage 5 – DNS-Based Data Exfiltration

Instead of exfiltrating data over HTTP, the malware encoded encrypted archive chunks directly into DNS TXT queries.

The malware constructed DNS requests in the following format:

```
xh.{machineId16}.{nonce10}.{sig12}.{chunkIdx}.{data}.bt.node.js
```

and:

```
xd.{machineId16}.{nonce10}.{sig12}.{chunkIdx}.{data}.bt.node.js
```

The malware also bypassed local enterprise DNS infrastructure by explicitly using public DNS resolvers:

```
const resolvers = [["1.1.1.1"], ["8.8.8.8"]];
```

This technique is particularly stealthy because:

- DNS traffic is often allowed outbound,
- organizations commonly monitor HTTP more closely than DNS,

- no active HTTP session is required.

The attacker-controlled authoritative nameserver reconstructed stolen data directly from DNS query logs.

Stage 6 – Detached Background Persistence

The malware attempted to continue execution independently of the original Node.js process by forking itself into a detached child process:

```
const child = childProcess.fork(
  path.resolve(moduleFilename),
  [],
  {
    detached: true,
    stdio: "ignore",
    windowsHide: true
  }
);
```

Copied

This allowed the payload to:

- survive parent process termination,
- suppress visible output,
- continue exfiltration silently in the background.

Infrastructure and Attribution

The attack infrastructure included:

Type	Value
C2 Domain	sh.azurestaticprovider.net
Domain	azurestaticprovider.net
Exfiltration Domain	bt.node.js
C2 IP	37.16.75.69

The domain `azurestaticprovider.net` was registered the same morning as the malicious package publication.

The attack leveraged a legitimate maintainer account:

atiertant

suggesting either:

- maintainer account compromise,
- insider abuse.

Exfiltration Hidden in Plain Sight

What makes this campaign especially notable is the malware's use of DNS as a covert exfiltration channel directly from inside a trusted npm package. In many recent supply chain attacks, stolen data is typically exfiltrated over HTTP or other more traditional command-and-control channels. In this case, the payload encrypted stolen credentials and infrastructure secrets, split them into small chunks, and embedded them directly inside DNS requests sent to attacker-controlled domains. This allowed the malware to hide data theft activity inside traffic patterns that many organizations may not inspect closely. The package itself also continued functioning normally throughout the infection process. There were no obvious crashes or visible indicators of compromise for developers using the dependency, allowing credential harvesting and background exfiltration to occur silently during normal application execution. Combined with targeted harvesting of cloud credentials, Kubernetes secrets, CI/CD artifacts, Terraform state, and developer tooling configurations, the campaign highlights the new operational baseline attackers are bringing to modern supply chain compromises.

Indicators of Compromise (IOCs)

Type	Value
Package	node-ipc@9.1.6
Package	node-ipc@9.2.3
Package	node-ipc@12.0.1
IP	37.16.75.69
Domain	sh.azurestaticprovider.net
Domain	azurestaticprovider.net
Domain	bt.node.js
DNS Pattern	*.bt.node.js
DNS Pattern	`^(xh
File Path	/tmp/nt-*/*.tar.gz
File Path	/tmp/nt-*/
File	node-ipc.cjs
Environment Variable	__ntw=1
Process Behavior	Detached node child process
DNS Behavior	Node.js TXT queries to 8.8.8.8 or 1.1.1.1
SHA-256	96097e0612d9575cb133021017fb1a5c68a03b60f9f3d24ebdc0e628d9034144
MD5	d1ba0419cb5e5de91b9b58e87b8322e1

Immediate Response Recommendations

Organizations should immediately:

- Remove affected package versions
- Treat impacted systems as potentially compromised
- Rotate all exposed credentials and tokens
- Rotate SSH keys
- Review CI/CD pipelines and secrets
- Investigate DNS logs for suspicious TXT queries
- Investigate outbound communication to listed infrastructure
- Validate software publishing systems and release workflows

Any developer workstation or CI/CD runner that executed the malicious package should be considered high risk.

How Upwind Can Help

Modern supply chain attacks increasingly abuse trusted developer tooling, CI/CD infrastructure, cloud identities, and runtime execution paths long before public signatures or retroactive threat intelligence become available.

Upwind helps organizations identify:

- malicious runtime package execution,
- suspicious Node.js process behavior,
- detached background child processes,
- unusual DNS TXT query activity,
- cloud credential access,
- CI/CD secret harvesting,
- and outbound communication to attacker-controlled infrastructure.

Upwind combines runtime visibility, identity context, network telemetry, and workload behavior analysis, organizations that allows us to identify active exploitation and real infrastructure impact rather than only detecting vulnerable packages after public disclosure.

The Supply Chain Strikes Again: Credential-Stealing Malware Hidden in node-ipc



Research

By Avital Harel & Lidor Machluf

May 19, 2026

Executive Summary Upwind identified a critical supply chain compromise involving durabletask==1.4.1, 1.4.2, and 1.4.3, three consecutive malicious releases of Microsoft's Azure Durable Task Python SDK published to PyPI. The malicious release contains a lightweight dropper embedded directly into durabletask/init.py. On import, the package downloads and executes a remote payload named rope.pyz from attacker-controlled infrastructure. The [...]



The New Face of Supply Chain Attacks

npm Malware Built for CI/CD and Cloud Compromise



Research

The New Face of Supply Chain Attacks: npm Malware Built for CI/CD and Cloud Compromise

By Avital Harel & Lidor Machluf

May 19, 2026

Executive Summary Upwind is tracking an active software supply chain campaign impacting multiple npm packages commonly used across developer tooling, frontend frameworks, CI/CD pipelines, and cloud-native application environments. We identified malicious payloads designed specifically to target CI/CD systems, cloud identities, GitHub credentials, npm publishing workflows, developer machines, and AI developer tooling. The campaign includes install-time [...]



Shai-Hulud: Here We Go Again

Dissecting a Supply Chain Worm
Across the TanStack Ecosystem



Research

Shai-Hulud: Here We Go Again – Dissecting a Supply Chain Worm Across the TanStack Ecosystem

By Avital Harel & Lidor Machluf

May 11, 2026

Executive Summary A new wave of the Mini Shai-Hulud campaign compromised dozens of official `@tanstack/*` npm packages by abusing CI/CD publishing workflows and trusted npm release mechanisms. Unlike traditional dependency malware focused only on downstream execution, this operation behaves as a self-propagating supply chain worm designed to continuously spread across repositories, developer environments, and CI/CD [...]



Welcome

Looking to create a secure cloud environment to accelerate your business?

Download our CNAPP Guide

For more information about how Upwind processes personal data, please refer to our Privacy Policy.